Journal of Engineering and Applied Sciences 13 (19): 7772-7782, 2018
ISSN: 1816-949X

Development for Remote Calculation for Smart Grid Curriculum:System Programming Approach

[1]Jeong-Ho Bak and [2]Jun-Ho Huh
[1]Department of Computer Engineering, Pukyong National University, Daeyeon, Busan, Republic of Korea
[2]Department of Software, Catholic University of Pusan, Busan, Republic of Korea

Abstract: In this study, the curriculum for smart grid remote reading was developed. When the character string formula is input in the client and it is transferred to server, the server receives and calculates character string sent from client and sends it to the client again. This study proposes a curriculum for the direct implementation of Shell and command is action in the system programming for smart grid remote reading for communication using message q and shared memory for creation of PIC calculation and for the implementation of remote calculator using socket. The infix notation calculator using stack was brought through the header file and the result value was converted and transferred to character string using sprint. In the further study, the performance of education by combining it to the middle and high school curriculum will be measured.

Key words: Remote calculation, smart grid, micro grid, curriculum, programming, demand response for micro grid

INTRODUCTION

As the threat of the nuclear power plant is accelerated due to the earthquake in Pohang, the smart grid has become a key issue. The smart grid is an intelligent power grid and the power rate calculation is very important (Huh and Seo, 2017; Dubey and Tomar, 2017; Pullagujju, 2016; Gururaj, 2016). However, textbooks and studies on this issue are very insufficient in Korea. Accordingly, this study proposes a curriculum to understand Shell command interpretation function action in the system programming and to execute the command interpretation function which is a representative action, for smart grid remote calculation (Huh and Seo, 2017; Arora *et al*., 2017; Bansal and Shricastava, 2017; Kickmeier-Rust and Albert, 2013).

To understand file system inside the system LS-ALR action was embodied which is a command of Linux. As in the existing action, file type, file authority, number of hard link, user name, group name, size, date, time and file name are output in sorting when the command is input and it also, outputs sub-directories as well as external files (Brut *et al*., 2008; Floridi, 1999; Huh and Seo, 2014; Huh *et al*., 2016; Ngu and Huh, 2017; Hoic-Bozic *et al*., 2009).

The child process is generated using fork formula and the equation is calculated in the child process which is received from the parent process. And the result value is delivered to the parent process. The main function used here is msgsnd() function and the child process produces a result value by changing the infix notation to postfix notation (Ma and Zhou, 2000; Varga, 1999; Aydogmus and Aydogmus, 2009; Mason *et al*., 2013).

To understand a socket programming, the remote calculator is implemented using TCP/UDP. When character string formula is input in the client and sent to server the server calculates character string from client and then sends it to client again. The calculator structure used in the calculation is a infix notation calculator model using stack and it is set to receive data by converting to character string using sprinf.

Design of implementation: To implement of Shell, a first implementation detail the function is figured out as follows:

- Command interpreter function
- Programming function
- User configuration function

Representative functions of shell: The 3 representative functions of Shell and description of each function is as follows. Firstly, command interpreting function performs interpreter and translator role that interprets and delivers command between the user and the kernel. It interprets command input by user and command read from file and then runs a suitable program. Secondly, programming function is a function that creates a program as there is a

Corresponding Author: Jun-Ho Huh, Department of Software, Catholic University of Pusan, Geumjeong-Gu, 57 Oryundae-Ro, Busan, Republic of Korea (ROK)

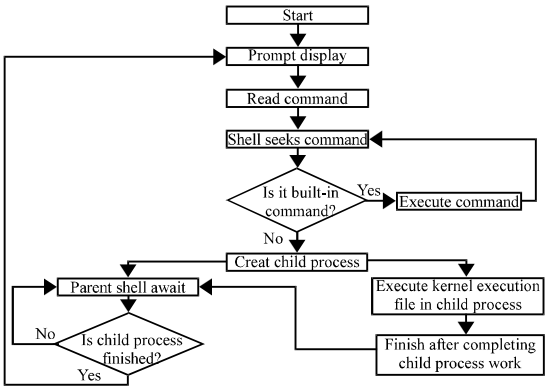

Fig. 1: Operating flow chart of shell

Table 1: File type separator

Character	File type
-	General file
a	Directory
b	Block device special file
C	Letter device special file
I	Symbolic link

programming function inside Shell. Thirdly, user configuration function configures user environment using initialization file function of Shell. It can create user environment characteristics by user such as path set, basic right set for new file and various environmental settings in the initialization file.

Figure 1 is an operating flowchart of Shell that is a process of interpreting and running command. To implement LS-ALR command, the prior knowledge as below is required. The details are as follows. Firstly, it needs to know the results when command is input for implementation.

Algorithm basic 'LS -ALR' command action:

```
drwxr-xr-x  2 root root 4096 2015-10-13 20:17 .
drwxr-xr-x. 11 root root 4096 2015-11-01 03:05 ..
-rw-r--r--. 1 root root   64 2015-10-13 02:33 Makefile
-rwxr-xr-x  1 root root 6948 2015-10-13 20:13 Myran
-rw-r--r--. 1 root root  608 2015-10-13 20:13 Myran.c
-rw-r--r--  1 root root  542 2015-10-13 19:59 Myran.c-
-rw-r--r--. 1 root root   39 2015-10-13 19:16 Myran.h
-rw-r--r--. 1 root root 2312 2015-10-13 20:13 Myran.o
```

When command is input as in Algorithm file type, file authority, number of hard link, user name, group name, size, date, time and file name produced. And at the same time, it is sorted in the order of file name and it is output in the same expression method with the sub-directory.

Table 1 The is a separator for file type and it must be considered when implementation. File type search related macro:

- Function
- If true, FIFO file
- If true, letter device special file

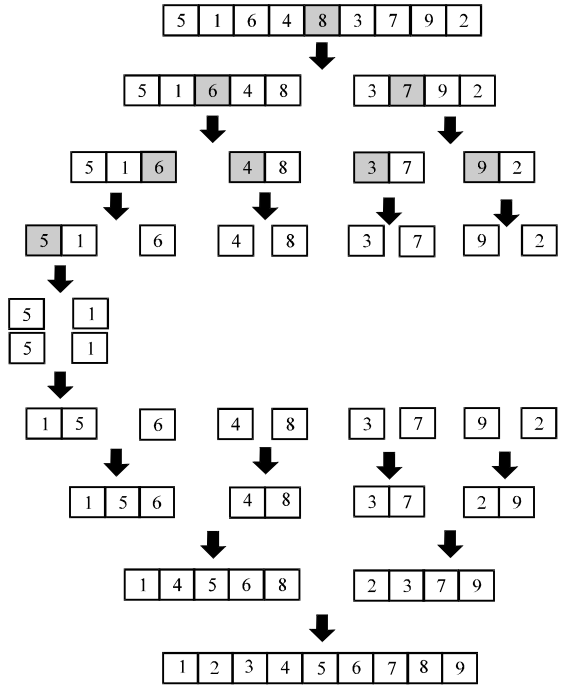

Fig. 2: Merge sorting structure

- If true, directory
- If true, block device special file
- If true, general file
- If true, symbolic link file
- If true, socket file

The macro names is a collection of macros used when file type separator is output. Example of user authority:

$$\text{Ex)st\_mode and } (\text{S\_IREAD} >> 3)$$

An example related to the user access right implementation. The access authority of group and other users than owner is carried out by moving the value of st_mode to 3 bit to left and performing AND by moving the constant value to 3 bit to right. The information is output in sorting and the sorting method used here will be a merge sorting. The related theory is as follows. The merge sorting is a sorting method that divides whole elements into one unit and then merges the divided element again. Complexity is nlogn which is most efficient among various sorting methods. Figure 2 is a merge sorting structure and the sorting order is as follows:

- Divide the sorted data set in half
- If the divided sub data set size is more than 2, 1 is repeated to this sub data set

- Combine 2 sub data sets that are from the same set and make one data set. When merging it, the element of data set is sorted according to the order.
- Repeat 3 until the data set becomes 1

Merge sorting method. This method will be used to output file list in this study.

MATERIALS AND METHODS

Proposed system: Before realizing Shell the function must be clarified. The goal is as follows:

- If the user input command, the relevant command is run
- If the user does not input suitable command, it outputs error
- If suitable command is executed, child process is created at parent process through fork. And in the child process, the program is executed according to the input command at the parent process through s exec function group
- Parent process waits until child process is finished
- Run cd and ls (function added)
- Input commands are recorded in log file, i.e., .txt file

Specific plan is as in specific plan for Shell implementation:

- Step 1: Parent process designates infinite loop as while (1)
- Step 2: Recognition of command entered to parent process: Use scanf to save command and option in the form of pointer, respectively
- Step 3: Implement Ls, copy process through fork function and deliver command option (suitable command recognition inspection and action check)
- Step 4: Cd implementation
- Step 5: Log file record implementation through time function citme and fpirntf function

As in the overall action method is that parent process is rotated to infinite loop which is branched to fork () according to command entered to parent process in order to implement relevant process.

Figure 3 is an implementation of step 1 and 2. Yellow box indicates the storage method of character string using a token. Orange box indicates the last of character string by rotating replicator I on the loop together. And it designates NULL in the last of character string according to the rule of existing exec function group. In summary, parent process is rotated to infinite loop using while and it was planned to save command and

```
//--------커맨드 인식부분-----//

char command[10];//커맨드 입력창
char seps[]=" ";//토큰용 구분자
char *token;

char *argv[50];//커맨드 전달용
int i=0;

printf("Park_Shell: ");
gets(command);//명령어를 인식합니다

token=strtok(command,seps);
while(token!=NULL)
{
        argv[i++]=token;
        token=strtok(NULL,seps);
}
argv[i]=NULL;//마지막에 NULL을 넣어준다.
//---------커맨드 인식부분 끝--//
```

Fig. 3: Step 1 and 2 implementation

```
switch(pid=fork())
{
        case -1:
        {
                printf("Error\n");
                exit(1);
                break;
        }
        case 0:
        {
                if(execv("/bin/ls",argv)==-1)
                {
                        printf("ls Error");
                        exit(1);
                }
                break;
        }
}
```

Fig. 4: Check option delivery to implement step 3

each option in the form of pointer from user through scanf function. But as the command and path must be separated based on space bar the study has to use the 'token' library. Strtok function prototype:

$$\text{char*strtok}(\text{char*str},\text{const*delimiters})$$

The is a function that divides character string and str is character string sort. And delimiters are sort that saves the separators. In the directly implemented mysh.c, seps[] sort is a separator sort.

As in Fig. 4, the execution part is made temporarily and is was executed in Shell in order to check action in step 2. Figure 5 is confirmed its action by implementing is. Option command is delivered but the following problems take place. Firstly, parent process does not wait for finish of child process. Second, it runs when other command is used than the designated command. Figure 6 is code added to solve previously mentioned problems.

```
Park_Shell: ls -al
Park_Shell: 합계 20
drwxr-xr-x. 2 root root 4096 2015-11-13 23:45 .
drwxr-xr-x. 4 root root 4096 2015-11-13 20:23 ..
-rw-r--r--. 1 root root  853 2015-11-13 23:45 mysh.c
-rwxr-xr-x. 1 root root 7425 2015-11-13 23:45 test
```

Fig. 5: Result of execution with problem

```
case 0://자식 노드 분기점
{
        if(execv("/bin/ls",argv)==-1)
        {
                printf("ls Error");
                exit(2);
        }
        break;
}
default:
{
        while(wait(&status)!=pid)
        continue;
        break;
}
```

Fig. 6: Code creation for synchronization (step 3 implementation (1))

```
if(strcmp(argv[0],"ls")==0)
{

        if(execv("/bin/ls",argv)==-1)
        {
                printf("ls Error");
                exit(2);
        }
        break;
}
else
{
        printf("명령어를 다시 입력하세요\n");
        exit(1);
}
```

Fig. 7: Code creation for synchronization (step 3 implementation (2))

Using wait command, it made parent process to wait for the finish of child process. To clarify the designated command, it uses argv [0] as a separator which is customarily used when execve function is used as in (Fig. 7) in order to determine if sentence.

Figure 8 is a result of previously mentioned problems implemented through the process of Fig. 6 and 7. To implement step 4, cd the process is copied as in the previous method and directory is changed in the copied process. Then, the result of cd called from child process at parent process is not reflected. As cd command is built-in command of Shell, if cd command is created as a separate program, the order becomes cd command implementation -> creation of child process(cd) in Shell ->

```
drwxr-xr-x. 2 root root 4096 2015-11-14 15:05 .
drwxr-xr-x. 4 root root 4096 2015-11-14 01:18 ..
-rw-r--r--. 1 root root 1180 2015-11-14 15:05 mysh.c
-rwxr-xr-x. 1 root root 7775 2015-11-14 02:24 test
[Park_Shell]:ls -al
합계 20
drwxr-xr-x. 2 root root 4096 2015-11-14 15:05 .
drwxr-xr-x. 4 root root 4096 2015-11-14 01:18 ..
-rw-r--r--. 1 root root 1180 2015-11-14 15:05 mysh.c
-rwxr-xr-x. 1 root root 7775 2015-11-14 02:24 test
[Park_Shell]:as
명령어를 다시 입력하세요
[Park_Shell]:fa
명령어를 다시 입력하세요
[Park_Shell]:
```

Fig. 8: Designated problem solving and result of step 3 implementation

```
//-----------cd구현 부분---------------//
if(strcmp(argv[0],"cd")==0)
{
        chdir(argv[1]);
        continue;
}
else if(strcmp(argv[0],"pwd")==0)
{
        if(argv[1]!=NULL)
        {
                printf("잘못된 명령어 입니다\n");
                continue;
        }
        else
        {
                getcwd(buff_for_pwd,100);
                printf("현재 디렉토리: %s\n",buff_for_pwd);
                continue;
        }
}
//-----------------------------------//
```

Fig. 9: Step 4 (cd, pwd) implementation

```
[Park_Shell]:pwd
현재 디렉토리: /Park/My_sh
[Park_Shell]:cd /Park
[Park_Shell]:cd /
[Park_Shell]:pwd
현재 디렉토리: /
[Park_Shell]:
```

Fig. 10: Cd and pwd implementation result

directory change at child process. As in this manner, if the child process is returned the directory of Shell is not changed.

Figure 9 is a code created to solve the above problem and the method is as follows. Cd function is inserted to the main directly and cd is run according to *argv[0] character string and 'continue' is inserted to exceed fork() function.

Figure 10 is a result to implement and run step 4 and the mentioned problems are all solved. Meanwhile, step 5 is to create execution command and time recording. Figure 11 is to create function that records log file and the text file to record opens in 'a' type. (If there is no file, open in 'w' mode), uses time and ctime function.

```
void Write_log(char *command)
{
        time_t t;
        time(&t);//기록하는 시간을 저장

        FILE *fp;

        fp=fopen("Park_log.txt","a");
        if(fp==NULL)//a모드로 개방 실패시...
        {
                fp=fopen("Park_log.txt","w");
                if(fp==NULL)
                {
                        printf("파일 개방에 실패하였습니다\n");
                        return;
                }
        }

        fprintf(fp,"%s : %s",command, ctime(&t));
        fclose(fp);
}
```

Fig. 11: Log file recording function

```
Park_log.6.txt
pwd : Sat Nov 14 22:29:01 2015
exit : Sat Nov 14 22:29:04 2015
```

Fig. 12: Ctime function prototype

```
void Write_log(char *command)
{
        time_t t;
        time(&t);//기록하는 시간을 저장

        FILE *fp;

        fp=fopen("/Park/My_sh/Park_log.txt","a");
        if(fp==NULL)//a모드로 개방 실패시...
        {
                fp=fopen("/Park/My_sh/Park_log.txt","w");
                if(fp==NULL)
                {
                        printf("파일 개방에 실패하였습니다\n");
                        return;
                }
        }

        fprintf(fp,"%-20s : %s",command, ctime(&t));
        fclose(fp);
}
```

Fig. 13: Problem solving and step 5 implementation

Algorithm ctime function prototype:

#include <time. h>
char *ctime (const time_t *clock)

In ctime function is a function that converts sec. unit time to a type that is easy to see by human and to return it to character string. It is output in day, month, day, hour, min, sec and year.

When Fig. 12 is executed, the result of Fig. 13 can be obtained and it shows log file is saved. But there are two things to consider based on the above results. First when command record starts by finishing and running program with exit input, it is not created in connection with the existing file but log file is created at the top path of execution file (In the Fig. 12, exit is input once and there is no further record). Second, log file needs to be

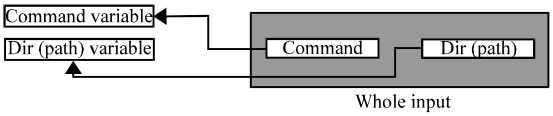

Fig. 14: LS -ALR implementation drawing

sorted and output but the length per sentence is different and the legibility is poor. To create high quality program two problems must be solved.

To solve the above problems, Fig. 14 code is created. When file is open, it is designated that the log file is created in a specific path. To enhance legibility of log file, it uses printf option, i.e., conversion character string option. The description of option used here is as follows. Code used in problem solving; "%-20 sec" mark used in means left sort and the number '20' means '20 columns are secured and used.

Implementation of LS-ALR command: As mentioned in the Introduction, now Linux command LS-ALR is implemented. The goal for implementation is as follows:

- Input must be operated when it is input with 2 (dir) format
- Output value-file type, file authority, no. of hard link, user name, group name, size, date, time and file name
- Output must be sorted out in the file name order. LS-ALR implementation goal

The implementation goal is as in and each function is implemented using the resource structure theory. When receiving input from keyboard, command part and dir part must be separated. Thus, when implementing with program, it is designed to implement in the method in Fig. 15.

Meanwhile, Fig. 16 is implemented using C in the input part based on design. Figure 17 shows sort to output file information. Remaining revision time or other part are created in additional sort, so that, it implements the time_t content is brought in the stat structure file.

As in Fig. 18, the information on type and authority of files in sub directory is obtained. Based on such information, the file type and authority must be clarified. For the clarification, the method used in the above is a method to use constant and macro in the required knowledge part.

Figure 19 and 20 are set of if sentence to clarify files. These sets save file separators into sort based on if sentence. S_ISDIR (mode) (((mode) and 0×4000) True (when it is directory).

```
char command[50];
char dirname[10];

scanf("%s",&command);
if(strcmp(command,"ll")!=0){

  puts("command error");
    return -1;}

scanf("%s",&dirname);

if(DirSeek(dirname)==-1)
{
        puts("실패");
        return -1;
}

free(dirname);
return 0;
```

Fig. 15: Input part implementation

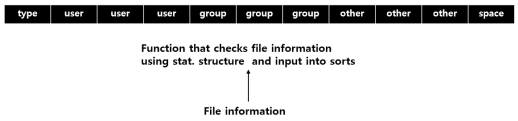

Fig. 16: Sort to output file information

In the clarification between file and directory, if condition is (Fig. 9), for example. It is implemented to hang if sentence simply and to input separators into sort such as 'd' and 'I' according to the result value. Total implementation step is a sum of file block unit other than file quantity as in Fig. 21-23:

- Function name: Func_Dir (file directory)
- Printf (finle information output)
- Determine whether there is sub director or not
- Func_Dir (File directory)

Implementation drawing for printing sub directory. To design sub directory, the regression method as in the above figure is used. The finish point of regression is the 'determination of whether there is sub directory or not.

Connection list is used as a resource structure to output information of each file. It serves to contain stat structure information in the Info_st in the structure defined earlier. At the same time, it inserts node and outputs information of file by deleting node of connection list. The result of program created by summing all designs in this study is as in Fig. 24. Figure 25 is a comparison photo between implemented command and existing command.

```
drwxr-xr-x. 2 root root
```

Fig. 17: Check existing function through is command

```
void Inspection_file(struct stat *buf, Info_st * info)
{
  //dir
        if(S_ISDIR(buf->st_mode))
                info->permission[PER0]='d';
        else if(S_ISLNK(buf->st_mode))
                info->permission[PER0]='l';
        else if(S_ISCHR(buf->st_mode))
                info->permission[PER0]='c';
        else if(S_ISBLK(buf->st_mode))
                info->permission[PER0]='b';
        else if(S_ISSOCK(buf->st_mode))
                info->permission[PER0]='s';
        else if(S_ISFIFO(buf->st_mode))
                info->permission[PER0]='P';
        else
                info->permission[PER0]='-';
```

```
//userMOD
if(buf->st_mode & S_IRUSR)
        info->permission[PER1]='r';
else
        info->permission[PER1]='-';
if(buf->st_mode & S_IWUSR)
        info->permission[PER2]='w';
else
        info->permission[PER2]='-';
if(buf->st_mode & S_IXUSR)
        info->permission[PER3]='x';
else if(buf->st_mode & S_ISUID)
        info->permission[PER3]='s';
else
        info->permission[PER3]='-';
```

Fig. 18: File separating function implementation (1)

```
//user
if(buf->st_mode & S_IROTH)
        info->permission[PER7]='r';
else
        info->permission[PER7]='-';
if(buf->st_mode & S_IWOTH)
        info->permission[PER8]='w';
else
        info->permission[PER8]='-';

if(buf->st_mode & S_IXOTH) //stiky
{
        if(buf->st_mode & S_ISVTX)
                info->permission[PER9]='t';
        else
                info->permission[PER9]='x';
}
else
{
        if(buf->st_mode & S_ISVTX)
                info->permission[PER9]='T';
        else
                info->permission[PER9]='-';
}

info->permission[PER10]='\0';
```

```
//group
if(buf->st_mode & S_IRGRP)
        info->permission[PER4]='r';
else
        info->permission[PER4]='-';
if(buf->st_mode & S_IWGRP)
        info->permission[PER5]='w';
else
        info->permission[PER5]='-';
if(buf->st_mode & S_IXGRP)
        info->permission[PER6]='x';
else if(buf->st_mode & S_ISGID)
        info->permission[PER6]='s';
else
        info->permission[PER6]='-';
```

Fig. 19: File separating function implementation (2)

```
합계 56
-rw-r--r--. 1 root root  5388
-rw-r--r--. 1 root root  6584
-rw-r--r--. 1 root root   144
-rwxr-xr-x. 1 root root 12231
-rw-r--r--. 1 root root  9841
-rw-r--r--. 1 root root  2602
-rw-r--r--. 1 root root  2376
-rw-r--r--. 1 root root  1822
```

Fig. 20: Example to implement total part

```
List list;
ListElmt * elmt={'\0'};
Info_st * info;
DIR * dirpt;
struct dirent * entry;
struct stat buf;
struct group * grp;
struct passwd * pwd;
struct tm * time;
char fileName[NAME_MAX];
int readc=0, total=0;
char * dirPath;

list_init(&list,DeleteInfo);
```

Fig. 21: Implementation of outlet using connection list (1) is a set of variables to implement outlet

```
while((entry=readdir(dirpt))!=NULL)
{
        strcpy(fileName, dirname);
        strcat(fileName, "/");
        strcat(fileName,entry->d_name);
        if((lstat(fileName,&buf)==0))
        {
                info=(Info_st *)malloc(sizeof(Info_st));
                pwd=getpwuid(buf.st_uid);
                grp=getgrgid(buf.st_gid);
                strcpy(info->userid, pwd->pw_name);
                strcpy(info->groupid, grp->gr_name);

                info->linkcount=buf.st_nlink;
                info->size=buf.st_size;

                time=localtime(&buf.st_mtime);

                info->date[0]=(time->tm_mon)+1;
                info->date[1]=time->tm_mday;
                info->time[0]=time->tm_hour;
                info->time[1]=time->tm_min;
```

Fig. 22: Implementation of outlet using connection list (2)

```
[root@localhost TT]# ./Testing
ll /
/:
합계 128
drwxr-xr-x13 root root0 11월  1 09:21 sys
dr-xr-xr-x2 root root12288 10월  4 13:16 sbin
drwxr-xr-x2 root root4096  9월 23 20:50 srv
drwxrwxrwt34 root root4096 11월  1 03:34 tmp
drwx------3 root root4096 10월  3 22:04 .dbus
drwxr-xr-x24 root root4096 10월  3 22:00 var
drwxr-xr-x2 root root4096 10월  5 01:12 temp
drwxr-xr-x2 root root4096  9월 23 20:50 mnt
dr-xr-xr-x175 root root0 11월  1 09:21 proc
drwxr-xr-x18 root root3820 11월  1 00:21 dev
dr-xr-xr-x30 root root4096 11월  1 00:21 .
drwxr-xr-x7 root root0 11월  1 09:21 selinux
drwxr-xr-x2 root root4096 10월 17 02:43 mywork
drwxr-xr-x4 root root1024 10월 25 02:29 home
drwxr-xr-x3 root root4096 11월  1 00:22 media
drwxr-xr-x117 root root12288 11월  1 00:55 etc
drwxr-xr-x2 root root4096 11월  1 02:40 다운로드
dr-xr-xr-x5 root root1024 10월  3 22:02 boot
dr-xr-xr-x30 root root4096 11월  1 00:21 ..
dr-xr-xr-x11 root root4096 10월  3 22:08 lib
dr-xr-x---27 root root4096 11월  1 13:54 root
drwxr-xr-x2 root root4096 10월 12 20:41 test
```

Fig. 23: Result of final implementation

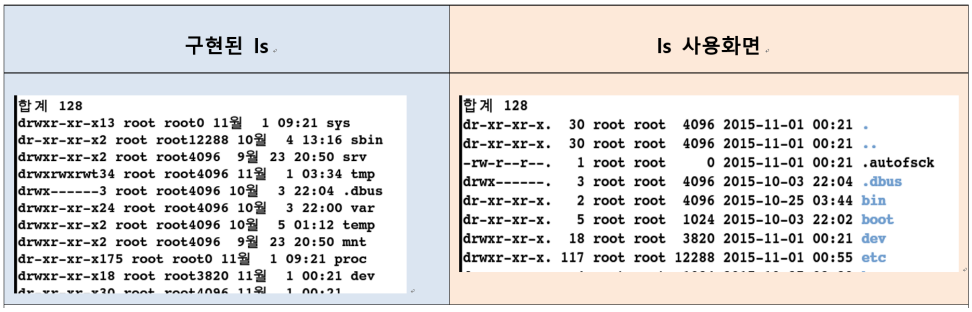

Fig. 24: Comparision between existing is command and implemented is command

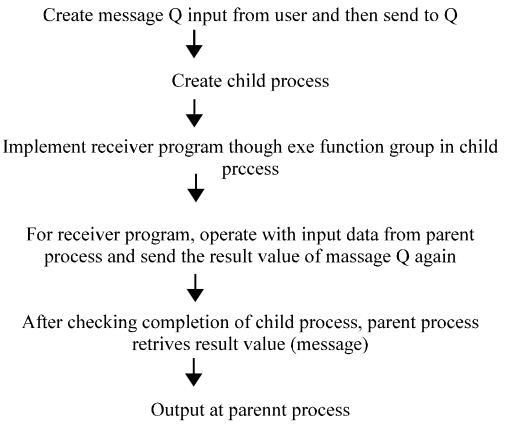

Fig. 25: Implementation plan

```c
#include<sys/msg.h>
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
#include<unistd.h>

struct mymsgbuf{
        long mtype;
        char mtext[80];
};

char* input_line()
{
        char* indata=(char*)malloc(sizeof(char)*80);

        printf("식을 입력하세요:");
        scanf("%s",indata);
        return indata;
}
```

Fig. 26: Input function implementation

RESULTS AND DISCUSSION

Communication using message q and shared memory process communication using message q: Figure 26 is an implementation plan for communication using message Q and the operating function will be added by the start of communication environment development.

Figure 27 is a detail that implements function input data from user. Figure 28 displays the creation to deliver message at parent process before creation of child process. And the message buffer of mesg structure is

```c
int main(void)
{
        key_t key;
        int msgid;
        struct mymsgbuf mesg;
        key=ftok("keyfile",1);
        msgid=msgget(key,IPC_CREAT|0644);
        if(msgid==-1)
        {
                perror("msgget");
                exit(1);
        }
        mesg.mtype=1;
        strcpy(mesg.mtext,input_line());//copy message

        if(msgsnd(msgid,(void*)&mesg,80,IPC_NOWAIT)==-1)//send message
        {
                perror("msgsnd");
                exit(1);
        }
```

Fig. 27: Message sending impementation befor fork function call

```c
        //--프로세스 분기점--//
        pid_t pid;
        int status;
        switch(pid=fork())
        {
                case -1:
                {
                        perror("fork");
                        exit(1);
                        break;
                }
                case 0://child process//
                {
                        if(execv("./receiver",(char*)NULL)==-1)
                        {
                                printf("자식 프로세스 생성 실패\n");
                                exit(1);
                        }
                        exit(0);//정상 종료
                        break;
                }
```

Fig. 28: Child process creation and program execution

```c
                default://부모프로세스//
                {
                        while(wait(&status)!=pid)
                        {continue;}

                        //----부모 프로세스 수신부---------//
                        struct mymsgbuf inmsg;
                        int len;

                        printf("부모프로세스에서 출력합니다\n");
                        len=msgrcv(msgid,&inmsg,80,0,0);
                        printf("%s\n",inmsg.mtext);
                        break;
                }
        }
```

Fig. 29: Data receiving part of parent process

designated to input return value of input function. In Fig. 29, child process is created and receiver program is executed and implemented.

```c
int main(void)
{
        struct mymsgbuf inmsg;
        key_t key;
        int msgid,len;

        key=ftok("keyfile",1);
        if((msgid=msgget(key,0))<0)
        {
                perror("msgget");
                exit(1);
        }
        len=msgrcv(msgid,&inmsg,80,0,0);
        printf("%s를 받았습니다\n",inmsg.mtext);

        //------------메세지 수신부 끝-----------//

        if(msgsnd(msgid,(void*)&inmsg,80,IPC_NOWAIT)==-1)//재전송//
        {
                printf("자식 프로세스에서 메세지 재 전송 실패\n");
                perror("msgsnd");
                exit(1);
        }

        return 0;
}
```

Fig. 30: Receiver program that seves reciving and re-sending

```
식을 입력하세요:1+9
1+9를 받았습니다
부모프로세스에서 출력합니다
1+9
[root@localhost MQ]#
```

Fig. 31: Communication configuration check

```c
#include"InfixCalculator.h"
struct mymsgbuf{
        long mtype;
        char mtext[80];
};

int main(void)
{
        struct mymsgbuf inmsg;
        key_t key;
        int msgid,len;

        key=ftok("keyfile",1);
        if((msgid=msgget(key,0))<0)
        {
                perror("msgget");
                exit(1);
        }
        len=msgrcv(msgid,&inmsg,80,0,0);
        printf("자식프로세스 에서 식 %s 를 받았습니다\n",inmsg.mtext);
        printf("연산을 시작합니다\n");
        //------------메세지 수신부 끝-----------//

        //-----------------식 계산---------------//
        int result=EvalInfixExp(inmsg.mtext);
        sprintf(inmsg.mtext,"%d",result);

        if(msgsnd(msgid,(void*)&inmsg,80,IPC_NOWAIT)==-1)//재전송//
        {
                printf("자식 프로세스에서 메세지 재 전송 실패\n");
                perror("msgsnd");
                exit(1);
        }
```

Fig. 32: Receiver that adds operating function

The receiver program saves the result value in the message again. So, the final value is to receive from parent process. It is implemented that the parent process after completing child process which is default, retrieves the message that the receiver program uploads (Fig. 30 and 31).

The receiver program receives message (parent process message), saves it to buffer send the details of buffer to Q again using msgsnd and sends to Q again. If the above codes are all input and implemented, the result of Fig. 32 can be obtained.

```
[root@localhost MQ]# ./sender
식을 입력하세요:(1+8)*9
자식프로세스 에서 식 (1+8)*9 를 받았습니다
연산을 시작합니다
------부모프로세스에서 출력합니다------
81
[root@localhost MQ]#
```

Fig. 33: Result of communication using message Q

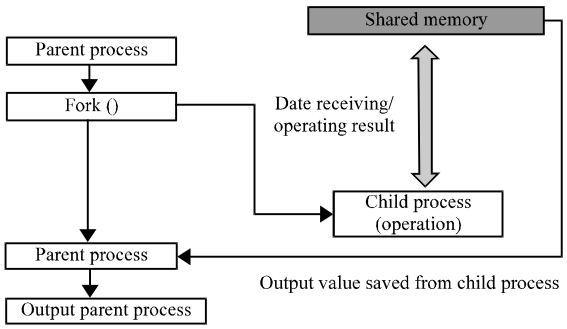

Fig. 34: Implementation plan

As in the existing requirements, calculator is used for operating receiver program with stack (infix notation is changed to postfix notation. Then, formula is calculated). Before saving the result value in inmsg, the integer is changed to character string to save message. Please note that Linux does not support itoa function sprintf function prototype:

- Sprintf(char*dest, “conversion character string”, change subject)

To solve it, use sprinf as in to serve the itoa function. When the above processes are gone through, the result of Fig. 33 and 34 can be obtained. In the child process, the operating formula from parent process is input and it is operated. Then, it is sent to parent process again.

Communication implementation using shared memory: Figure 35 is an aerial view of process communication implementation plan using shared memory.

Use input_line (return of input value from user) established before with strcpy function in the shared memory and send character string before going to child process. And clear memory in order to use shared memory of child process.

As in Fig. 36, after checking the finish of child process, parent process retrieves the result value saved in the shared memory in the child process and outputs it.

```
char* input_line()
{
        printf("전송할 메세지 입력:");
        char* indata=(char*)malloc(sizeof(char)*20);
        scanf("%s",indata);
        return indata;
}

int main()
{
        int shmid,i;
        char *shmaddr, *shmaddr2;
        shmid=shmget(IPC_PRIVATE,20,IPC_CREAT|0644);
        if(shmid==-1)
        {
                perror("shmget");
                exit(1);
        }
        //--------식 입력 및 전송----------//
        shmaddr2=(char*)shmat(shmid,(char*)NULL,0);
        printf("====Parent Process====\n");
        strcpy(shmaddr2,input_line());//식을 전송합니다
        shmdt((char*)shmaddr2);//메모리 해제
        printf("\n");
```

Fig. 35: Communication implementation using shared memory (formula input and send)

```
    case 0:
    {
            int res;//결과 값을 저장할 변수
            sleep(2);
            shmaddr=(char*)shmat(shmid,(char*)NULL,0);
            printf("====Child Process====\n");
            printf("%s 식을 메모리에서 가져옵니다 \n",shmaddr);//데이터 수신

            res=EvalInfixExp(shmaddr);
            sprintf(shmaddr,"%d",res);//재전송
    //      strcpy(shmaddr,"Resend!");
            printf("입력 된 식 연산 후 재 전송\n\n");
            shmdt((char*)shmaddr);
            exit(0);
            break;
    }
```

Fig. 36: Communication implementation using shared memory (formula input and send)

```
[root@localhost shm]# ./shard
====Parent Process====
전송할 메세지 입력:(3+7)*(1+9)

====Child Process====
(3+7)*(1+9) 식을 메모리에서 가져옵니다
입력 된 식 연산 후 재 전송

====Parent Process====
100
[root@localhost shm]#
```

Fig. 37: Implementation result

After going through the above processes, the result of Fig. 37 can be obtained. Save formula in the shared memory at parent process and retrieve formula in the shared memory at child process. Then, save it in the shared memory. Afterwards, parent process retrieves and outputs result value. In the LS-ALR command implementation part the below problems can be checked:

- No sort is made based on file name
- Code complexity is enhanced by the use of connection list

The code complexity can reduce the quantity of complex cod by using C++ STL instead of C. If using more advanced sort algorithm the quicker result can be output in listing and sorting multiple files. As each module is separated, the user can modify algorithm according to situation. A curriculum for the smart grid remote meter reading technology course is being introduced in this study once a character string is entered by the client and transferred to the server, it will be sent back to the client after undergoing necessary calculation. The curriculum includes direct implementation of Shell and command Is operations in system programming for remote calculation function of the smart grid, communication and IPC calculator compilation using message queue and shared memory as well as implementation of a remote calculator using the socket. A stack-oriented infix notation calculator was brought along through the header file just as it was and the resulting value will be transmitted after it has been transformed into a string. The performance of the curriculum in the secondary and higher educations will be evaluated in the future research.

CONCLUSION

As the smart grid is an intelligent power grid, the power rate calculation is very important. However, there are limited textbooks and studies in Korea. Accordingly, this study proposed a curriculum for smart grid remote calculation. In the smart grid remote power calculation implementation, it can be applied to the production of software that processes complex formula data remotely. And it would be a good example for undergraduate students who attempt to understand differences of TCP/UDP. If Q is introduced to process in sequence by receiving input of various formulas instead of stack used for implementation, it can process various operations sequentially. And if code is improved with C++, it adds thread theory and produces results at the same time. As the operating part is separated with function, if another algorithm is turned to function for addition, it would add flexibility in the efficiency for complex calculation.

ACKNOWLEDGEMENT

This research was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No.2017R1C1B5077157).

REFERENCES

Arora, N., M. Martolia and A. Ashok, 2017. A comparative study of the image registration process on the multimodal medical images. Asia Pacific J. Convergent Res. Interchange, 3: 1-17.

Aydogmus, Z. and O. Aydogmus, 2009. A web-based remote access laboratory using SCADA. IEEE. Trans. Educ., 52: 126-132.

Bansal, M. and L. Shricastava, 2017. Performance analysis of wireless mobile adhoc network with different types of antennas. Asia Pacific J. Convergent Res. Interchange, 3: 33-44.

Brut, M., S. Buraga, S. Dumitriu, G. Grigoras and M. Girdea, 2008. A competency-oriented modeling approach for personalized E-learning systems. Proceedings of the 2008 3rd International Conference on Internet and Web Applications and Services, June 8-13, 2008, IEEE, Athens, Greece, ISBN:978-0-7695-3163-2, pp: 410-415.

Dubey, D. and G.S. Tomar, 2017. Echelon based pose generalization of facial images approaches. Asia Pac. J. Convergent Res. Interchange, 3: 63-75.

Floridi, L., 1999. Information ethics: On the philosophical foundation of computer ethics. Ethics Inf. Technol., 1: 33-52.

Gururaj, A., 2016. A study on mining user-aware uncommon consecutive topic patterns in report streams. Asia Pacific J. Convergent Res. Interchange, 2: 17-23.

Hoic-Bozic, N., V. Mornar and I. Boticki, 2009. A blended learning approach to course design and implementation. IEEE. Trans. Educ., 52: 19-30.

Huh, J.H. and K. Seo, 2014. Development of competency-oriented social multimedia computer network curriculum. J. Multimedia Inf. Syst. Korea Multimedia Soc., 1: 113-116.

Huh, J.H. and K. Seo, 2017. An indoor location-based control system using bluetooth beacons for IoT systems. Sens., 17: 1-22.

Huh, J.H., S. Otgonchimeg and K. Seo, 2016. Advanced metering infrastructure design and test bed experiment using intelligent agents: Focusing on the PLC network base technology for Smart Grid system. J. Supercomput., 72: 1862-1877.

Kickmeier-Rust, M.D. and D. Albert, 2013. Using hasse diagrams for competence-oriented learning analytics. Proceedings of the International Workshop on Human-Computer Interaction and Knowledge Discovery in Complex, Unstructured, Big Data, July 1-3, 2013, Springer, Berlin, Germany, ISBN:978-3-642-39145-3, pp: 59-64.

Ma, J. and D. Zhou, 2000. Fuzzy set approach to the assessment of student-centered learning. IEEE. Trans. Educ., 43: 237-241.

Mason, G.S., T.R. Shuman and K.E. Cook, 2013. Comparing the effectiveness of an inverted classroom to a traditional classroom in an upper-division engineering course. IEEE. Trans. Educ., 56: 430-435.

Ngu, H.C.V. and J.H. Huh, 2017. B+-tree construction on massive data with Hadoop. Cluster Comput., 1: 1-11.

Pullagujju, G.K., 2016. Identifying Trojan Facebook applications. Asia Pac. J. Convergent Res. Interchange, 2: 1-6.

Varga, A., 1999. Using the OMNeT++ discrete event simulation system in education. IEEE. Trans. Educ., 42: 1-11.