lect10 HW

시스템프로그래밍 1분반 12180626 성시열

1) Compile and run mysh.c in section 5. What is the difference between mysh and the system shell(the login shell that runs when you log in)?

```c
#include <stdio.h>
#include <unistd.h>
#include <string.h>

void main(){
        int x, y, status, i;
        char buf[50];
        char *argv[10];

        for(i=0; i<10;i++){
                printf("$");
                scanf("%s", buf);
                argv[0] = buf;
                argv[1] = 0;

                x = fork();
                if(x==0){
                        printf("I am child to execute %s\n", buf);
                        y = execve(buf, argv, 0);
                        if(y<0){
                                perror("exec failed");
                                exit(1);
                        }
                } else  wait(&status);
        }
}
~
~
```

```
[12180626@linuxer1 ~]$ vi mysh.c
[12180626@linuxer1 ~]$ gcc -o mysh mysh.c
mysh.c: In function 'main':
mysh.c:23:5: warning: incompatible implicit declaration of built-in function 'ex
it' [enabled by default]
[12180626@linuxer1 ~]$ mysh
$ex1
I am child to execute ex1
I am ex1
$ex2
I am child to execute ex2
I am ex1
$ex3
I am child to execute ex3
hello: 12602
hello: 0
```

주어진 코드를 작성해보니 실행파일이 정상적으로 작동되는 것을 확인할 수 있었다. 실제 shell과의 차이점은 for 조건문으로 인해 10번만 반복된다는 점, scanf로 값을 받으므로 공백이 포함된 명령문은 실행을 못한다는 점이 있다. ex1 ex2 ex3를 차례대로 실행시키고 있었는데 ex3 이후로 실행이 되지 않은 것을 확인하였다.

ctrl + c를 통해 정지한 뒤 vi ex3.c를 통해 코드를 확인해보았다.

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

void main(){
        int x;
        x = fork();
        printf("hello: %d\n", x);
        for(;;);
}
~
```

코드를 확인한 결과 for(;;);를 통해 무한루프에 빠졌음을 알 수 있다.

2) Find all ancestor processes of your login shell.

```
[12180626@linuxer1 ~]$ mysh
$/bin/ps -ef
I am child to execute /bin/ps
  PID TTY          TIME CMD
13267 pts/4    00:00:00 bash
14026 pts/4    00:00:00 mysh
14028 pts/4    00:00:00 ps
$I am child to execute -ef
exec failed: No such file or directory
$$myexec
I am child to execute myexec
[/home/sp5/12180626]$ /bin/ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 Jun15 ?        00:00:10 /usr/lib/systemd/systemd --switc
root         2     0  0 Jun15 ?        00:00:00 [kthreadd]
root         3     2  0 Jun15 ?        00:00:01 [ksoftirqd/0]
root         5     2  0 Jun15 ?        00:00:00 [kworker/0:0H]
root         7     2  0 Jun15 ?        00:00:00 [kworker/u:0H]
root         8     2  0 Jun15 ?        00:00:00 [migration/0]
```

(출력결과가 너무 길어 일부 생략하였습니다.)

shell의 모든 상위 프로세스를 찾기 위해 ps –ef를 실행시켜 보았다. mysh은 scanf로 명령어를 받으므로 띄어쓰기를 인식하지 못하여 /bin/ps 까지만 실행된 것을 알 수 있다.
mysh의 pid는 14026임을 확인하였다. 전체 프로세스를 확인하기 위해 앞서 제작한 myexec를 활용하였다. myexec를 통해 /bin/ps –ef를 입력한 결과 모든 상위 프로세스의 정보를 출력하였다.

```
root         1     0  0 Jun15 ?        00:00:10 /usr/lib/systemd/systemd --switc
root      1020     1  0 Jun15 ?        00:00:22 /usr/sbin/sshd -D
root     13264  1020  0 15:18 ?        00:00:00 sshd: 12180626 [priv]
12180626 13266 13264  0 15:18 ?        00:00:00 sshd: 12180626@pts/4
12180626 13267 13266  0 15:18 pts/4    00:00:00 -bash
12180626 14026 13267  0 15:59 pts/4    00:00:00 mysh
12180626 14089 14026  0 16:02 pts/4    00:00:00 myexec
12180626 14090 14089  0 16:02 pts/4    00:00:00 /bin/ps -ef
```

모든 상위 프로세스를 직접 찾은 결과이다.

3) (Builtin Command) Improve mysh such that it exits when the user types "exit".

```c
#include <stdio.h>
#include <unistd.h>
#include <string.h>

void main(){
        int x, y, status, i;
        char buf[50];
        char *argv[10];

        for(i=0; i<10;i++){
                printf("$");
                scanf("%s", buf);
                argv[0] = buf;
                argv[1] = 0;

                if(strcmp(buf, "exit")==0){
                        printf("shell is finish\n");
                        exit(1);
                }

                x = fork();
                if(x==0){
                        printf("I am child to execute %s\n", buf);
                        y = execve(buf, argv, 0);
                        if(y<0){
                                perror("exec failed");
                                exit(1);
                        }
                } else  wait(&status);
        }
}
~
```

```
[12180626@linuxer1 ~]$ mysh
$ex1
I am child to execute ex1
I am ex1
$exit
shell is finish
[12180626@linuxer1 ~]$
```

기존 mysh 코드에서 exit가 입력되었을 때, "shell is finish"라는 문자열을 출력한 뒤 exit()를 실행한다. 컴파일하고 실행한 결과, ex1을 실행시킬 수 있고, exit를 입력함과 동시에 "shell is finish"가 출력되고, mysh을 빠져나온 것을 알 수 있다.

4) Improve mysh further such that it can handle a command with arguments, such as "/bin/ls -l". Use gets() or fgets() to read the command.

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


void main(){
        int x, y, status, i;
        char buf[50];
        char *argv[10];
        for(i=0; i<10;i++){
                printf("$");
                gets(buf);

                int j;
                argv[0] = strtok(buf, " ");
                for(j = 1; ; j++){
                        argv[j] = strtok(NULL, " ");
                        if(argv[j] == NULL) break;
                }
                argv[j] = 0;

                if(strcmp(buf, "exit")==0){
                        printf("shell is finish\n");
                        exit(1);
                }

                x = fork();
                if(x==0){
                        printf("I am child to execute %s\n", buf);
                        y = execve(buf, argv, 0);
                        if(y<0){
                                perror("exec failed");
                                exit(1);
                        }
                } else  wait(&status);
        }
}
~
~
```

옵션까지 받기 위해 공백까지 입력 받을 수 있는 gets()를 사용한다. gets()로 값을 입력받은 뒤, strtok를 통해 공백을 기준으로 argv에 하나씩 넣어준다.

```
[12180626@linuxer1 ~]$ vi mysh.c
[12180626@linuxer1 ~]$ gcc -o mysh mysh.c
mysh.c: In function ‘main’:
mysh.c:13:3: warning: ‘gets’ is deprecated (declared at /usr/include/stdio.h:640
) [-Wdeprecated-declarations]
[12180626@linuxer1 ~]$ mysh
$/bin/cat f1
I am child to execute /bin/cat
12180626
$/bin/cat f2
I am child to execute /bin/cat
12180626
$exit
shell is finish
[12180626@linuxer1 ~]$
```

컴파일 후 실행해본 결과, cat f1, cat f2와 같이 공백이 포함된 명령어도 잘 실행되는 것을 알 수 있다.

5) (Handling &) Change the shell such that it can handle '&' at the end of the command.

명령어 마지막 부분에 &를 넣을 시에 shell은 기다리지 않고 즉시 다음 prompt를 인쇄하고 다음 사용자 명령을 기다린다.

즉, 부모인 경우에 wait()을 통해 자식이 실행되고 종료될 때까지 기다렸지만, & 문자를 받게 되면 기다리지 않고 prompt에서 다음 사용자의 명령을 기다리게 한다.

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

void main(){
        int x, y, status, i;

        for(i=0; i<10; i++){
                char buf[50];
                char *argv[10];
                char *tmp = 'a';
                //
                printf("$");
                gets(buf);
                //
                if(strcmp(buf, "exit")==0){
                        printf("shell is finish\n");
                        exit(1);
                }
                //
                argv[0] = strtok(buf, " ");

                int j;
                for(j = 1; ; j++){
                        argv[j] = strtok(NULL, " ");
                        if(argv[j] == NULL) break;
                }
                argv[j] = 0;
                //
                if(strcmp(argv[j-1], "&")==0){
                        tmp = argv[j-1];
                        argv[j-1] = 0;
                }
                //
                x = fork();
                if(x==0){
                        printf("I am child to execute %s\n", buf);
                        y = execve(argv[0], argv, 0);
                        if(y<0){
                                perror("exec failed");
                                exit(1);
                        }
                }
                //
                else{   //child's process
                        if(strcmp(tmp,"&")!=0){
                                wait(&status);
                        }
                }
        }
}
```

이를 위해 else와 if를 활용한다. 원래는 x!=0(else)에서 자식 프로세스일 때 무조건 wait() 통해 부모 프로세스가 실행되고 종료되길 기다렸다면, 명령어 끝에 '&'가 아닐 때만 wait()을 역할을 수행한다.

```
#include <stdio.h>

void main(){
        printf("I am ex1\n");
        for(;;);
}
~
~
```

컴파일 후 실행하기 전 ex1파일에 미리 무한루프를 포함시켰다.

```
[12180626@linuxer1 ~]$ mysh
$ex1 &
$I am child to execute ex1
I am ex1
/bin/ps
I am child to execute /bin/ps
  PID TTY          TIME CMD
13267 pts/4    00:00:00 bash
15728 pts/4    00:01:15 ex1
15738 pts/4    00:00:00 mysh
15739 pts/4    00:00:18 ex1
15765 pts/4    00:00:00 ps
Segmentation fault (core dumped)
```

mysh를 실행시킨 뒤 ex1 &를 입력한 결과 출력문이 실행되고 무한루프에 빠진 것을 알 수 있다. 이 때 /bin/ps을 실행시켜 현재 실행되고 있는 process를 확인하였다.

"Segmentation fault (core dumped)"는 허가되지 않은 메모리에 접근할 때 발생하는 오류인데 어느 부분에 접근할 때 발생하였는지 파악하지 못하였다.

6) (Handling relative path) Make your shell handle relative paths assuming the executable file always exists in /bin directory. When the user enters only the command name (e.g. "ls -l", "cp f1 f2", etc), build a full path such as "/bin/ls", "/bin/cp", etc. and perform exec. Use sprintf() to build the full path.
프로그램 실행 후 문자열을 입력할 때, /bin/을 입력하지 않더라도 명령어가 실행될 수 있도록 하는 프로그램을 구성한다. 이 때 sprintf()를 사용한다.

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

void main(){
        int x, y, status, i;

        for(i=0; i<10; i++){
                char buf[50];
                char s_buf[50];
                char *argv[10];
                char *tmp = '0';
                //
                printf("$");
                gets(buf);
                //
                if(strcmp(buf, "exit")==0){
                        printf("shell is finish\n");
                        exit(1);
                }
                sprintf(s_buf, "%s%s", "/bin/", buf);
                //
                argv[0] = strtok(s_buf, " ");

                int j;
                for(j=1;;j++){
                        argv[j] = strtok(NULL, " ");
                        if(argv[j] == NULL) break;
                }
                argv[j] = 0;
                //
                x = fork();
                if(x==0){
                        printf("I am child to execute %s\n", buf);
                        y = execve(argv[0], argv, 0);
                        if(y<0){
                                perror("exec failed");
                                exit(1);
                        }
                }
                else{   //child's process
                        if(strcmp(argv[j-1], "&")==0)
                                argv[j-1] = 0;

                        else //tmp!="&"
                                wait(&status);
                }
        }
}
```

s_buf 문자열을 추가하여 "/bin/"을 입력하지 않아도 자동으로 포함되는 문자열을 만들어준다. 이후 s_buf를 활용하여 strtok 함수를 적용한다.
또한, 위에서 발생한 "Segmentation fault (core dumped)"를 부모 프로세스에서만 두가지 경우로 적용된다는 것을 알고 수정한 결과 오류까지 해결할 수 있었다.

```
[12180626@linuxer1 ~]$ mysh
$ls
I am child to execute ls
cphw4     ex1.c   ex3      ex5     f1      f9       mycat2    mysh
cphw4.c   ex11    ex3.c    ex5.c   f1.c    f91      mycat2.c  mysh.c
d1        ex11.c  ex33     ex6     f1out   f92      mycat3    myxxd
d2        ex13    ex33.c   ex6.c   f2      f93?     mycat3.c  myxxd.c
d3        ex13.   ex4      ex7     f3      fabc     mycp      newhw4.c
echo      ex13.c  ex4.c    ex7.c   f4      hw33     mycp.c    outfiles
ex.8      ex2     ex44     ex8     f5      hw4      myecho    sw2.wav
ex0       ex2.c   ex44.c   ex8.c   f6      hw4.c    myecho.c  swvader03.wav
ex0.c     ex22    ex444    ex9     f7      mycat    myexec    x
ex1       ex22.c  ex444.c  ex9.c   f8      mycat.c  myexec.c  y.c
$cat f1
I am child to execute cat f1
12180626
$cat f2
I am child to execute cat f2
12180626
$cat f3
I am child to execute cat f3
12180626
$cat f4
I am child to execute cat f4
12180626
$cat f5
I am child to execute cat f5
12180626
$cat f6
I am child to execute cat f6
12180626
$cp f1 f7
I am child to execute cp f1 f7
$cat f1
I am child to execute cat f1
12180626
$cat f7
I am child to execute cat f7
12180626
[12180626@linuxer1 ~]$
```

코드를 작성하고 컴파일 한 뒤 실행하였다.
원하는대로 "/bin/"을 작성하지 않더라도 명령어를 실행할 수 있었고, for문을 통해 10번을 반복하므로, 10번의 명령어를 실행한 뒤 mysh을 탈출한다.

6-1) Use getenv("PATH") to retrieve PATH environment variable and use strtok() to extract each system path. Display each system path line by line.

/usr/lib64/ccache
/usr/local/bin
/usr/bin
.....

```
[12180626@linuxer1 ~]$ echo $PATH
/usr/lib64/ccache:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:.:/home/sp5/
12180626/.local/bin:/home/sp5/12180626/bin:.
```

먼저 $ echo $PATH를 통해 어떻게 출력되는지 확인해보았다. 줄바꿈을 ":"을 기준으로 해야 하는 것을 알 수 있다.

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>

void main(){
        char *paths[500];
        int i, j;
        char *x = getenv("PATH");
        char env[500];
        //printf("%s\n",x);
        sprintf(env, "%s", x);

        paths[0] = strtok(env, ":");
        for(i = 1; ;i++){
                paths[i] = strtok(NULL, ":");
                if(paths[i] == NULL) break;
        }
        paths[i] = 0;

        for(j = 0; j < i; j++){
                printf("%s\n", paths[j]);
        }
}
~
~
```

getenv("PATH")를 통해 위 출력값과 같은 한 줄로 늘어진 PATH를 x에 저장한다.
getenv는 return 값을 포인터변수로만 받을 수 있는데 x의 값을 포인터변수에서 배열로 변경하기 위해 sprintf()를 사용한다. 배열형태로 변경한 이유는 ":"를 기준으로 strtok를 사용하여 tokenize를 하기 위해서이다. tokenize 된 문자열들을 paths[]에 저장하고 for문을 통해 paths[]를 줄바꿈하며 한 인덱스씩 출력한다.

```
[12180626@linuxer1 ~]$ vi path.c
[12180626@linuxer1 ~]$ gcc -o path path.c
[12180626@linuxer1 ~]$ path
/usr/lib64/ccache
/usr/local/bin
/usr/bin
/usr/local/sbin
/usr/sbin
.
/home/sp5/12180626/.local/bin
/home/sp5/12180626/bin
.
```

컴파일 후 실행한 결과 한 줄로 출력되던 PATH가 ":"를 기준으로 줄바꿈 된 것을 확인할 수 있다.

7) (Handling relative path) Change the shell such that it can handle relative path for the command in general. The shell will search the PATH environment variable to compute the full path of the command when it is given as a relative path name. Use getenv("PATH") to obtain the pointer to the value of the PATH environment variable. Note you need to copy the string of the PATH variable into another char array before you start extracting each path component with strtok() since strtok() destroys the original string.

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

void main(){
	int x, y, status, i;

	for(i=0; i<10; i++){
		char buf[50];
		char s_buf[50];
		char *argv[10];
		char *tmp = '0';
		char arr[100];

		char *paths[500];
		char *z = getenv("PATH");
		char env[500];
		sprintf(env, "%s", z);
		//
		paths[0] = strtok(env, ":");
		int k;
		for(k=1;;k++){
			paths[k] = strtok(NULL, ":");
			if(paths[k] == NULL) break;
		}
		paths[k]=0;
		//
		printf("$");
		gets(buf);
		//
		if(strcmp(argv[0], "exit")==0){
			printf("shell is finish\n");
			exit(1);
		}
		//
		//
		argv[0] = strtok(buf, " ");
		int j;
		for(j=1;;j++){
			argv[j] = strtok(NULL, " ");
			if(argv[j] == NULL) break;
		}
		argv[j]=0;
		//
		x = fork();
		if(x==0){
			if(argv[0][0] == '/') { //absolute path
```

```
                        printf("I am child to execute %s\n", buf);
                        y = execve(argv[0], argv, 0);
                        if(y<0){
                                perror("exec failed");
                                exit(1);
                        }
                }
                else { //
                        printf("I am child to execute %s\n", buf);
                        int m;
                        for(m=0; m<k; m++){
                                sprintf(arr, "%s%s%s", paths[m], "/", argv[0]);
                                argv[0] = arr;
                                y = execve(argv[0], argv, 0);
                                if(y<0){
                                        continue;
                                }
                        }
                }
        }
        else {
                if(strcmp(argv[j-1], "&")==0)
                        argv[j-1] = 0;

                else
                        wait(&status);
        }
    }
}
```

6-1에서 getenv("PATH")로 PATH의 환경 변수를 받아오고 strtok()로 " : " 마다 끊어 추출하였던 것을 이용하여 명령어를 상대 경로로 입력 받았을 때 PATH의 환경 변수에서 입력 받은 명령어가 자신의 경로에 맞는 것을 찾아 실행하도록 shell을 수정했다.
5번 코드에서 "/bin/"의 경로를 사용하는 명령어만 수행 가능했는데 위처럼 shell을 수정해 줌으로써 PATH의 환경 변수에 있는 모든 경로들의 명령어를 사용할 수 있게 되었다.
paths에 저장된 PATH경로 중 가장 마지막에 들어가는 경로를 5번 코드의 "/bin/"대신 넣어 줌으로써 마지막 경로의 명령어를 실행한다.
자식 프로세스가 실행되는 중에 입력받은 값이 절대경로일 때와 상대경로일 때를 구분하는데, 절대경로일 때는 기존에 입력받은 값을 그대로 복제하고 실행하면 된다.
그렇지 않을 때는 paths에 저장된 PATH 경로를 하나씩 arr이라는 문자열에 저장하고, argv[0]에 넣어준 뒤, 절대경로를 입력할 때처럼 복제한다.

```
[12180626@linuxer1 ~]$ vi mysh.c
[12180626@linuxer1 ~]$ gcc -o mysh mysh.c
mysh.c: In function 'main':
mysh.c:13:15: warning: initialization makes pointer from integer without a cast
[enabled by default]
mysh.c:29:3: warning: 'gets' is deprecated (declared at /usr/include/stdio.h:640
) [-Wdeprecated-declarations]
[12180626@linuxer1 ~]$ mysh
$ls
Segmentation fault (core dumped)
```

컴파일 후 실행한 결과 첫 명령어를 입력하자마자 "Segmentation fault (core dumped)" 오류가 발생하였다. "Segmentation fault (core dumped)"는 허가되지 않은 메모리에 접근할 때 발생하는 오류인데 어느 부분에 접근할 때 발생하였는지 파악하지 못하였다.