TI DSP, MCU, Xilinx Zynq FPGA Based Programming Expert Program

Instructor – Innova Lee(Sanghoon Lee)
gcccompil3r@gmail.com
Student – Hyungju Kim
mihaelkel@naver.com

How to use signal function

```
#include <signal.h>

typedef void (*sighandler_t)(int);

sighandler_t signal(int signum, sighandler_t handler);
```

signal function has 2 parameters, int signum, sighandler_t handler.
sighandler_t is a function pointer, has prototype of void (*)(int).
in the below program, parent process will print a series of numbers, 1 to 10000, at intervals of 0.05sec.
when the child process exits, it returns a signal. then, signal(SIGCHLD, my_sig) will execute.
as a result of that, signal function acts like interrupt.

```c
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>

void term_status(int status){
    if(WIFEXITED(status))
        printf("(exit)status : 0x%x\n",WEXITSTATUS(status));
    else if(WTERMSIG(status))
        printf("(signal)status : 0x%x, %s\n",
                status & 0x7f, WCOREDUMP(status) ? "core dumped" : "");
    while(1);
}

void my_sig(int signo){
    int status;
    wait(&status);
    term_status(status);
}

int main(void){
    pid_t pid;
    int i;

    signal(SIGCHLD, my_sig);
    if((pid = fork()) > 0)
        for(i=0;i<10000;i++){
            usleep(50000);
            printf("%d\n",i+1);

        }
    else if(pid == 0)
        sleep(5);
    else{
        perror("fork()");
        exit(-1);
    }
    return 0;
}

```

execlp funcion

```
#include <unistd.h>

extern char **environ;

int execl(const char *path, const char *arg, ...
                /* (char  *) NULL */);
int execlp(const char *file, const char *arg, ...
                /* (char  *) NULL */);
int execle(const char *path, const char *arg, ...
                /*, (char *) NULL, char * const envp[] */);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
int execvpe(const char *file, char *const argv[],
                char *const envp[]);
```

exe function executes the file in the parameter.

look at the below program. You would think after\n will be printed after execlp function executes. but, if you check the result, you can see "after\n" is not printed. as fork() function create a child process, exe function change the process. so, all the codes in the previous process will not execute.

```
1  #include <unistd.h>
2  #include <stdio.h>
3  int main(void){
4       execlp("ps","ps","-e","-f",NULL);
5
6       printf("after\n");
7       return 0;
8  }
9
```

Colored by Color Scripter

exe function without change the process

How can we use exe function without change the process?

the answer is, create the child process first, and change that like below.

```
1     #include <unistd.h>
2     #include <stdio.h>
3     #include <sys/wait.h>
4     int main(void){
5          int status;
6          pid_t pid;
7          if((pid = fork()) > 0){
8               wait(&status);
9               printf("prompt >");
10         }
11         else if(pid == 0)
12              execlp("ps","ps","-e","-f",NULL);
13         return 0;
14
15    }
16
```

Colored by Color Scripter

Kill process

If you run the below program, time information be printed every 1 second. check the process by using ps -ef |grep.

```c
#include <stdio.h>
#include <sys/types.h>
int main(void){
	while(1){
		sleep(1);
		system("date");
	}
	printf("after\n");
	return 0;

}

```

```
howard@ubuntu:~/HomeworkBackup/23th$ ps -ef | grep a.out
howard     2548  2441  0 01:21 pts/18   00:00:00 ./a.out
howard     3073  2441  0 03:30 pts/18   00:00:00 ./a.out
```

You can check the process(a.out) is running. then, do the same thing after close the terminal.

```
howard@ubuntu:~/HomeworkBackup/23th$ ps -ef | grep a.out
howard     3281  3081  0 03:31 pts/4    00:00:00 grep --color=auto a.out
```

can see like above, the process has exited. but service program need to continue running, when it is closed. daemon process does.

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
int daemon_init(void){
	int i;
	if(fork() > 0)
		exit(0);
	setsid();
	chdir("/");
	umask(0);
	for(i=0;i<64;i++)
		close(i);

	signal(SIGCHLD, SIG_IGN);
	return 0;
}

int main(void){
	daemon_init();
	for(;;);
	return 0;
}

```

It kills parent process with the option that ignore the signal of it's child process and umask(0).

```
howard@ubuntu:~/HomeworkBackup/23th$ ps -ef | grep a.out
howard     3337  1420 95 03:40 ?        00:00:10 ./a.out
howard     3350  3339  0 03:41 pts/18   00:00:00 grep --color=auto a.out
```

if you want to kill daemon process, just type "kill -9 pid"