

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

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



## **System Programming**

#### Implement Is -R option

-R option is for showing all files included in all lower directories. To implement it, You can use recursive call.

```
howard@ubuntu: ~/HomeworkBackup/22th
 1 #include <sys/types.h>
 2 #include <stdio.h>
 3 #include <unistd.h>
 4 #include <dirent.h>
 5 #include <sys/stat.h>
 6 #include <string.h>
 8 void recursive_dir(char* dname);
10 int main(int argc,char* argv[]){
       recursive_dir(" ");
12
       return 0;
13 }
15 void recursive_dir(char* dname){
       struct dirent* p;
       struct stat buf;
18
       DIR* dp;
19
       chdir(dname);
       dp = opendir(".");
20
21
22
23
24
25
26
       printf("\t%s :\n",dname);
       while(p = readdir(dp))
            printf("%s\n",p->d_name);
       rewinddir(dp);
       while(p = readdir(dp)){
           stat(p->d_name,&buf);
27
28
29
           //shorcut(2nd if) , strcmp returns 0 when it is true.
           if(S_ISDIR(buf.st_mode))
                if(strcmp(p->d_name,".") && strcmp(p->d_name,".."))
30
                    recursive dir(p->d name);
       chdir("...");
       closedir(dp);
34 }
```

#### fork() function

This is for creating a new process, the same as origin process, called "child process". It has return value. The value is pid for child process. If the parent has no child, return value be "0".

```
parent pid = 5763, cpid = 5764
howard@ubuntu:~/HomeworkBackup/22th$ child : pid = 5764, cpid = 0
```

Using fork function, we can prove the system can do context switching.

```
noward@ubuntu: ~/HomeworkBackup/22th
 1 #include <unistd.h>
 2 #include <stdio.h>
3 #include <errno.h>
 4 #include <stdlib.h>
 6 int main(void){
       pid_t pid;
       int i:
       pid = fork();
       if(pid > 0){
11
           while(1){
12
                for(i = 0; i < 26; i++){
13
                    printf("%c ",i + 'A');
14
                    fflush(stdout);
16
17
       else if(pid == 0){
18
19
           while(1){
20
                for(i = 0;i < 26;i++){
21
                    printf("%c ",i + 'a');
22
                    fflush(stdout);
23
24
25
       }
else{
26
           perror("fork() ");
28
           exit(-1);
29
30
       printf("\n");
       return 0;
```



## **System Programming**

#### fork() function

```
YzZaAbBcCdDeEfFgGhH<sup>.</sup>
uvwxyzaQbcRdSeTfUgVh
IqJrKsLtMuNvOwPxQyR:
CjkDlEmFnGoHpIqJrKsL
dWeXfYgZhAi^C
howard@ubuntu:~/HomeworkBackup/22th$
```

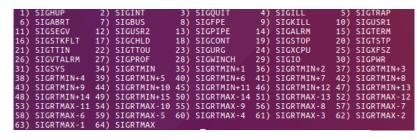
You can see the processes are executing alternately. that is, OS offers "context switching".

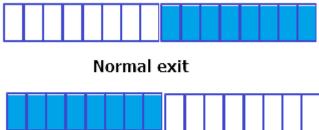
when child process is finished, parent should receive signal. But the parent couldn't receive(by reason of sleep etc..), child become defunct state.

```
3754 2166 0 Mar22 pts/18
                                     00:00:00 ./
howard
howard
         3755 3754 0 Mar22 pts/18
                                     00:00:00 ./
                                     00:00:00 ./
         5892 2166 0 03:12 pts/18
howard
                                     00:00:00
howard
         5893 5892 0 03:12 pts/18
                                                    ] <defunct>
                                     00:00:00 grep --color=auto
         5895 5874 0 03:12 pts/4
noward@ubuntu:~/HomeworkBackup/22th$
```

#### wait() function

it is waiting until the child process be finished. When the child process ends, it returns extract status. Below is the kill list (can check with kill –l instructor)





Abnormal exit

Normal extract status has upper 8bits, and abnormal has lower 8 bits. You can see the correct value by bit operating.

Normal extract : status >> 8 Abnormal extract : status & 0x7f ->last 1 bit(128) has other option.



## **System Programming**

#### Pipe communication

when implementing real-time communication system, we use non-block option. But, using multiple-process, we can make it with block.

```
howard@ubuntu: ~/HomeworkBackup/22th
 1 #include <unistd.h>
 2 #include <stdio.h>
 3 #include <errno.h>
 4 #include <stdlib.h>
 5 #include <fcntl.h>
7 int main(void)
       int fd, ret;
       char buf[1024];
       pid t pid;
       fd = open("myfifo",0_RDWR);
11
       if((pid = fork()) > 0){
12
           for(;;){
               ret = read(0, buf, sizeof(buf));
               buf[ret] = 0;
               printf("Keyboard : %s\n",buf);
17
18
       else if(pid == 0){
19
20
           for(;;){
               ret = read(fd, buf, sizeof(buf));
               buf[ret] = 0;
               printf("myfifo : %s\n",buf);
       else{
27
           perror("fork()");
28
           exit(-1);
29
       close(fd);
       return 0;
```

#### **Family process**

process has family pointer. pptr, cptr, yptr, optr, link(for priority queue).

