TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

2018.03.23
22 일차
강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com

학생 – 신민철
akrn33@naver.com

fork1.c

```
#include<stdio.h>
#include<unistd.h>
int main(void)
{
	printf("before\n");
	fork();//자신이 있는 시점 밑에를 복사한다.
	printf("after\n");
	return 0;
}
```

fork2.c

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

int main(void)
{
	pid_t pid;
	pid = fork();//자식의 pid 값을 반환함.
	if(pid >0)
	{
		printf("parent\n");
	}
	else if(pid ==0)
		printf("child\n");
	else
	{
		perror("fork()");//perror 은 어떤 에러났는지 알려주는 애
		exit(-1);
	}
```

```
	return 0;
}
```

fork3.c

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

int main(void)
{
	pid_t pid;
	pid = fork();
	if(pid > 0)//여기서는 부모프로세스니까 cpid 출력
		printf("parent : pid = %d, cpid = %d\n", getpid(), pid);
	else if(pid == 0)//여기서는 자식프로세스라서 cpid 는 출력 x
		printf("child : pid = %d, cpid = %d\n", getpid(), pid);
	else
	{
		perror("fork()");//에러처리함수 fork()의 에러를 알려줌
		exit(-1);
	}
	return 0;
}
```

fork4.c

//Context Switching 을 볼수있다. 프로세스가 두개 켜져있으니까 하나만 실행하지 않고 번갈아가면서 출력하는 것을 볼 수 있음.

```
#include<stdio.h>
```

```
#include<errno.h>
#include<stdlib.h>
#include<unistd.h>

int main(void)
{
	pid_t pid;
	int i;
	pid = fork();
	if(pid > 0)
	{
		while(1){
			for(i = 0; i < 26; i++)
				{
					printf("%c", i+ 'A');
					fflush(stdout);
				}
		}
	}
	else if(pid ==0)
	{	while(1){
			for(i = 0; i < 26; i++)
				{
					printf("%c", i + 'a');
					fflush(stdout);//오버플로우 안나게 하려고.
				}
		}
	}
	else
	{
		perror("fork()");
		exit(-1);
	}
	printf("\n");
	return 0;
```

```
}
```

fork9.c

```
#include<unistd.h>
#include<stdio.h>
#include<errno.h>
#include<stdlib.h>
#include<fcntl.h>

int main(void)
{
	pid_t pid;
	if((pid = fork()) > 0)
	{
		sleep(1000);//자식이 죽었는데 자고있어서 처리를 못함.
	}
	else if(pid == 0)
		;//자식프로세스인데 죽었음 근데 부모가 처리안해줘서 좀비가됨
	else
	{
		perror("fork()");
		exit(-1);
	}
	return 0;
}
```

ipc.c//자식프로세스에서 변수를 증가시키는 코드

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

int global = 100;
```

```
int main(void)
{
	int local = 10;
	pid_t pid;
	int i;
	pid = fork();
	if(pid > 0)
		printf("global : %d, local : %d\n",global,local);
	else if(pid ==0)
	{
		global++;
		local++;
		printf("global: %d, local: %d\n",global,local);
	}
	else
	{
		perror("fork()");
		exit(-1);
	}
	printf("\n");
	return 0;
}
```

lsmo9.c//디렉토리 안의 디렉토리 ,파일 그 안의파일 까지 다 보여주는 코드

```
#include<stdio.h>
#include<fcntl.h>
#include<sys/types.h>
#include<dirent.h>
#include<unistd.h>
#include<sys/stat.h>
#include<string.h>

void recursive_dir(char* dname);
```

```
int main(int argc, char* argv[])
{
	recursive_dir(".");
	return 0;
}

void recursive_dir(char* dname)
{
	struct dirent* p;
	struct stat buf;
	DIR* dp;
	chdir(dname);
	dp = opendir(".");
	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);
		if(S_ISDIR(buf.st_mode))
			if(strcmp(p->d_name,".")&&		strcmp(p-
>d_name,".."))
				recursive_dir(p->d_name);
	}
	chdir("..");
	closedir(dp);
}
```

sharevm.c

```
#include<stdio.h>

int main(void)
{
	int a = 10;
```

```
        printf("&a %#p\n", &a);
        sleep(1000);
        return 0;
}
```
//이 위에있는 변수의 주소를 다른프로세스가 접근하려하면 세그먼트 파울트 뜬다. 접근 못하게함.

Sharevm2.c

```
#include<stdio.h>

int main(void)
{
        int *p = 0x7ffc725a4e74;//위에있는 sharevm.c 의 주소인데
        printf("&a:%#p\n",*p);//접근한값을 출력하려고 하면
        return 0;//세그먼트파울트 뜸
}
```

wait.c

```
#include<stdio.h>
#include<fcntl.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<sys/wait.h>

int main(void)
{
        pid_t pid;
        int status;
        if((pid = fork()) > 0)
        {
                wait(&status);//기다렸다가 자식이 처리한요청 받음
        printf("status: %d\n",status);//자식이요청한거 출력해봄
        }
```

```
        else if(pid == 0)
                exit(7);//자식이 죽으면서 7 을 남김
        else
        {
                perror("fork()");
        exit(-1);
        }
        return 0;
}
```

wait3.c//자식이 요청한것을 비트연산해서 원래의 값으로 출력하는 코드

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

int main(void)
{
        pid_t pid;
        int status;
        if((pid = fork()) > 0 )
        {
                wati(&status);
                printf("status:0x%x\n",(status>>8)&0xff);
        }
        else if(pid == 0)
                exit(7);
        else
        {
```

```
		perror("fork()");
		exit(-1);
	}
	return 0;
}
```

wait4.c//자식이 죽으면서 남긴 값을 원래값으로 바꿀때 함수사용하는코드

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

int main(void)
{
	pid_t pid;
	int status;
	if((pid = fork()) > 0 )
	{
		wait(&status);
		printf("status:0x%x\n",WEXITSTATUS(status));
	}
	else if(pid == 0)
		exit(7);
	else
	{
		perror("fork()");
		exit(-1);
	}
	return 0;
}
```

wait5.c//정상 종료시 WEXITSTATUS 함수를 사용하면 나오는 값

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

int main(void)//정상종료이기때문에 0 이 출력된다.WEXITSTATUS 를 출력하는건 비정상종료
{
	pid_t pid;
	int status;
	if((pid = fork()) > 0)
	{
		wait(&status);
		printf("status:0x%x\n",WEXITSTATUS(status));
		printf("status:0x%d\n",status-128);
		printf("status:0x%x\n",status&0x7f);
	}
	else if(pid == 0)
		abort();//정상종료
	else
	{
		perror("fork()");
		exit(-1);
	}
	return 0;
}
```