응용과제 10

Makefile

```
# Makefile

cc = gcc
CFLAG = -g

all :
        make p_cook.exe
        make c_cook.exe

p_cook.exe : p_cook.c
        gcc -o p_cook.exe p_cook.c

c_cook.exe : c_cook.c
        gcc -o c_cook.exe c_cook.c
```

<소스코드>

```
# Makefile

cc = gcc
CFLAG = -g

all :
        make p_cook.exe
        make c_cook.exe

p_cook.exe : p_cook.c
        gcc -o p_cook.exe p_cook.c

c_cook.exe : c_cook.c
        gcc -o c_cook.exe c_cook.c
```

부모 프로세스 - 손님 접대 (주문 및 요리 제공)

SIGINT과 SIGUSR1시그널을 처리하기위한 핸들러

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

int pid_child; //자식의 pid를 할당하기 위한 전역변수 pid_child선언
/*********************************************************************/
void int_handler(int signo)
{
        //주문이 없는상태에서 SIGINT 또는 SIGQUIT시그널이 오는경우
        kill(pid_child, SIGINT); //자식종료
        printf("\n자식 프로세스를 종료하였습니다.\n");
        exit(0); //자신종료
}
/*********************************************************************/
void sigusr1_handler(int signo)
{
        //자식으로부터 SIGUSR1 시그널이 오면 요리 제공
        printf("---------- parent process ----------\n");
        printf("완성된 요리를 전달받아 손님에게 제공하였습니다.\n");

        struct sigaction act;
        sigemptyset(&act.sa_mask);
        act.sa_flags = 0;
        act.sa_handler = int_handler; //SIGINT가 오는경우 다시 int_handler로 동작
        if(sigaction(SIGINT, &act, (struct sigaction *)NULL) < 0)
        {
                perror("sigaction 에러 발생");
                exit(3);
        }
}
/*********************************************************************/
```

main함수

```
int main(void)
{

        pid_child = fork();

        if(pid_child < 0)
        {
                perror("fork 에러 발생 ");
                exit(1);
        }
        //자식프로세스인경우 exec로 프로그램의 몸체를 c_cook.exe로 변경
        else if(pid_child == 0)
        {
                if(execl("./c_cook.exe", "c_cook.exe", NULL) < 0)
                {
                        perror("execl 에러 발생 ");
                        exit(2);
                }
        }

        struct sigaction act;
        sigemptyset(&act.sa_mask);
        act.sa_flags = 0;
        act.sa_handler = int_handler;
        //주문전 SIGINT시그널 발생하는경우
        if(sigaction(SIGINT, &act, (struct sigaction *)NULL) < 0)
        {
                perror("sigaciton 에러 발생 ");
                exit(3);
        }

        while(1)        //반복해서 메뉴출력
        {
                printf("---------- parent process ----------\n");
                printf("음식을 주문하시겠습니까? (y/n) : ");
                char answer;
                scanf("%c", &answer);

                switch(answer)
                {
                        case 'Y' :
                        case 'y' : //주문이 들어오면 SIGUSR1을 제외한 모든시그널 블록
                                sigemptyset(&act.sa_mask);
                                act.sa_flags = 0;
                                act.sa_handler = sigusr1_handler;
                                sigaction(SIGUSR1, &act, NULL);
                                //사용자로부터 주문받아 자식프로세스에게 주문전달
                                if(sigaction(SIGUSR1, &act, (struct sigaction *)NULL) < 0)
                                {
                                        perror("sigaction 에러 발생 ");
                                        exit(3);
                                }

                                sigemptyset(&act.sa_mask);
                                act.sa_flags = 0;
                                act.sa_handler = SIG_IGN;
                                int signum;
                                for(signum=1; signum<=40; signum++) //SIGUSR1을 제외한 모든 시그널 무시
                                {
                                        if(signum != SIGUSR1)
                                        {
                                        sigaction(signum, &act, NULL);
                                        }
                                }

                                kill(pid_child, SIGUSR1); //자식에게 SIGUSR1 시그널 보냄
                                printf("주문을 요청하였습니다.\n");
                                pause();
                                break;
                        case 'N':
                        case 'n':
                                printf("주문프로그램을 종료합니다.\n");
                                exit(0);
                        default :
                                printf("잘못된 입력입니다. y or n\n");
                                exit(4);
                }
                getchar();
        }
}
```

p_cook.c 소스코드

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

int pid_child; //자식의 pid를 할당하기 위한 전역변수 pid_child선언
/*****************************************************************/
void int_handler(int signo)
{
        //주문이 없는상태에서 SIGINT 또는 SIGQUIT시그널이 오는경우
        kill(pid_child, SIGINT); //자식종료
        printf("₩n자식 프로세스를 종료하였습니다.₩n");
        exit(0); //자신종료
}
/*****************************************************************/
void sigusr1_handler(int signo)
{
        //자식으로부터 SIGUSR1 시그널이 오면 요리 제공
        printf("---------- parent process ----------₩n");
        printf("완성된 요리를 전달받아 손님에게 제공하였습니다.₩n");

        struct sigaction act;
        sigemptyset(&act.sa_mask);
        act.sa_flags = 0;
        act.sa_handler = int_handler; //SIGINT가 오는경우 다시 int_handler로 동작
        if(sigaction(SIGINT, &act, (struct sigaction *)NULL) < 0)
        {
                perror("sigaction 에러 발생");
                exit(3);
        }
}
/*****************************************************************/

int main(void)
{

        pid_child = fork();

        if(pid_child < 0)
        {
                perror("fork 에러발생");
                exit(1);
        }
        //자식프로세스인경우 exec로 프로그램의 몸체를 c_cook.exe로 변경
        else if(pid_child == 0)
        {
                if(execl("./c_cook.exe", "c_cook.exe", NULL) < 0)
                {
```

```
                perror("execl 에러 발생");
                exit(2);
        }
}

struct sigaction act;
sigemptyset(&act.sa_mask);
act.sa_flags = 0;
act.sa_handler = int_handler;
//주문전 SIGINT시그널 발생하는경우
if(sigaction(SIGINT, &act, (struct sigaction *)NULL) < 0)
{
        perror("sigaciton 에러 발생");
        exit(3);
}

while(1)        //반복해서 메뉴출력
{
        printf("---------- parent process ----------₩n");
        printf("음식을 주문하시겠습니까? (y/n) : ");
        char answer;
        scanf("%c", &answer);

        switch(answer)
        {
                case 'Y' :
                case 'y' : //주문이 들어오면 SIGUSR1을 제외한 모든시그널 블록
                        sigemptyset(&act.sa_mask);
                        act.sa_flags = 0;
                        act.sa_handler = sigusr1_handler;
                        sigaction(SIGUSR1, &act, NULL);
                        //사용자로부터 주문받아 자식프로세스에게 주문전달
                        if(sigaction(SIGUSR1, &act, (struct sigaction *)NULL) < 0)
                        {
                                perror("sigaction 에러 발생");
                                exit(3);
                        }

                        sigemptyset(&act.sa_mask);
                        act.sa_flags = 0;
                        act.sa_handler = SIG_IGN;
                        int signum;
                        for(signum=1; signum<=40; signum++) //SIGUSR1을 제외한 모든 시그널 무시
                        {
                                if(signum != SIGUSR1)
                                {
                                sigaction(signum, &act, NULL);
                                }
                        }
```

```
                        kill(pid_child, SIGUSR1); //자식에게 SIGUSR1 시그널 보냄
                        printf("주문을 요청하였습니다.₩n");
                        pause();
                        break;
                case 'N':
                case 'n':
                        printf("주문프로그램을 종료합니다.₩n");
                        exit(0);
                default :
                        printf("잘못된 입력입니다. y or n₩n");
                        exit(4);
        }
        getchar();
    }
}
```

자식 프로세스 – 조리 (요리 시작 – 완료)

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

#define TIME_FOR_COOK 3

void sigusr1_handler(int signo);


/*************************************************************************/
void alrm_handler(int signo)
{       //요리가 완성됨을 출력
        printf("완성된 요리를 전달합니다.\n");
        kill(getppid(), SIGUSR1); //부모에게 SIGUSR1시그널을 보냄

        struct sigaction act;
        sigemptyset(&act.sa_mask);
        act.sa_flags = 0;
        act.sa_handler = SIG_DFL;
        int signum;
        for(signum=1; signum<=40; signum++) //모든시그널 무시를 디폴트 동작으로 되돌림
        {
                sigaction(signum, &act, (struct sigaction *)NULL);
        }

        sigemptyset(&act.sa_mask);
        act.sa_flags = 0;
        act.sa_handler = sigusr1_handler;
        if(sigaction(SIGUSR1, &act, (struct sigaction *)NULL) < 0)
        {
                perror("sigaction 에러 발생");
                exit(1);
        }
}
/*************************************************************************/
void sigusr1_handler(int signo)
{
        printf("---------- child process ----------\n");
        printf("주문이 접수되었습니다.\n");

        struct sigaction act;
        sigemptyset(&act.sa_mask);
        act.sa_flags = 0;
        act.sa_handler = SIG_IGN;
        int signum;
        for(signum=1; signum<=40; signum++) //SIGALRM을 제외한 모든 시그널 무시
        {
                if((signum != SIGALRM) && (signum != 9) && (signum != 19))
                {
                        sigaction(signum, &act, (struct sigaction *)NULL);
                }
        }

        sigemptyset(&act.sa_mask);
        act.sa_flags = 0;
        act.sa_handler = alrm_handler;
        if(sigaction(SIGALRM, &act, (struct sigaction *)NULL) < 0)
        {
                perror("sigaction 에러발생");
                exit(1);
        }

        //ALRM시그널 받을경우 알람핸들러 호출
        if(sigaction(SIGALRM, &act, (struct sigaction *)NULL) < 0)
        {
                perror("sigaction 에러 발생");
                exit(1);
        }
        //일정 조리시간 후 SIGALRM시그널을 보냄(알람핸들러 호출)
        alarm(TIME_FOR_COOK);
}
/*************************************************************************/

int main(void)
{

        struct sigaction act;
        sigemptyset(&act.sa_mask);
        act.sa_flags = 0;
        act.sa_handler = sigusr1_handler;

        //부모로부터 SIGUSR1을 받아오면 조리시작(핸들러 호출)
        if(sigaction(SIGUSR1, &act, (struct sigaction *)NULL) < 0)
        {
                perror("sigaction 에러 발생");
                exit(1);
        }

        while(1)
        {
                sleep(999);     //시그널 발생까지 대기
        }

        return 0;
}
```

c_cook.c 소스코드

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

#define TIME_FOR_COOK 3

void sigusr1_handler(int signo);


/*************************************************************************/
void alrm_handler(int signo)
{       //요리가 완성됨을 출력
        printf("완성된 요리를 전달합니다.₩n");
        kill(getppid(), SIGUSR1); //부모에게 SIGUSR1시그널을 보냄

        struct sigaction act;
        sigemptyset(&act.sa_mask);
        act.sa_flags = 0;
        act.sa_handler = SIG_DFL;
        int signum;
        for(signum=1; signum<=40; signum++) //모든시그널 무시를 디폴트 동작으로 되돌림
        {
                sigaction(signum, &act, (struct sigaction *)NULL);
        }

        sigemptyset(&act.sa_mask);
        act.sa_flags = 0;
        act.sa_handler = sigusr1_handler;
        if(sigaction(SIGUSR1, &act, (struct sigaction *)NULL) < 0)
        {
                perror("sigaction 에러 발생");
                exit(1);
        }
}
/*************************************************************************/
void sigusr1_handler(int signo)
{
        printf("---------- child process ----------₩n");
        printf("주문이 접수되었습니다.₩n");

        struct sigaction act;
        sigemptyset(&act.sa_mask);
        act.sa_flags = 0;
        act.sa_handler = SIG_IGN;
        int signum;
        for(signum=1; signum<=40; signum++) //SIGALRM을 제외한 모든 시그널 무시
        {
```

```
                if((signum != SIGALRM) && (signum != 9) && (signum != 19))
                {
                        sigaction(signum, &act, (struct sigaction *)NULL);
                }
        }

        sigemptyset(&act.sa_mask);
        act.sa_flags = 0;
        act.sa_handler = alrm_handler;
        if(sigaction(SIGALRM, &act, (struct sigaction *)NULL) < 0)
        {
                perror("sigaction 에러발생");
                exit(1);
        }

        //ALRM시그널 받을경우 알람핸들러 호출
        if(sigaction(SIGALRM, &act, (struct sigaction *)NULL) < 0)
        {
                perror("sigaction 에러 발생");
                exit(1);
        }
        //일정 조리시간 후 SIGALRM시그널을 보냄(알람핸들러 호출)
        alarm(TIME_FOR_COOK);
}
/*************************************************************************/

int main(void)
{

        struct sigaction act;
        sigemptyset(&act.sa_mask);
        act.sa_flags = 0;
        act.sa_handler = sigusr1_handler;

        //부모로부터 SIGUSR1을 받아오면 조리시작(핸들러 호출)
        if(sigaction(SIGUSR1, &act, (struct sigaction *)NULL) < 0)
        {
                perror("sigaction 에러 발생");
                exit(1);
        }

        while(1)
        {
                sleep(999);      //시그널 발생까지 대기
        }

        return 0;
}
```

부모 프로세스 기능

1.부모 프로세스는 반복해서 메뉴를 출력
- while문을 사용하여 반복으로 출력해주었습니다.

2. 사용자로부터 주문을 받아 자식 프로세스에게 주문 내용 전달.(SIGUSR1)
- scanf를 사용하여 answer가 y 또는 Y인 경우 자식에게 SIGUSR1시그널을 보내 주문을 전달하였습니다.

3. 일단 주문을 받으면 SIGUSR1을 제외한 다른 시그널은 모두 무시
- sigaction함수를 사용하여 for루프문을 돌려 SIGUSR1을 제외한 시그널의 핸들러를 SIG_IGN으로 지정해줌으로써 다른시그널들을 무시하도록 동작하였습니다.

4. 자식으로부터 SIGUSR1(요리 완료)가 오면 화면에 출력 (요리 제공)
- pause함수로 SIGUSR1시그널이 올때까지 대기하다가 오는 경우 sigaction으로 sigusr1_handler가 처리하도록 해주었습니다. 화면 출력은 핸들러에서 동작합니다.

5. 주문이 없는 상태에서 SIGINT 또는 SIGQUIT 시그널이 들어오는 경우 자식을 먼저 종료시킨 후 자신도 종료
- int_handler를 작성하여 주문을 받기전 sa_handler로 int_handler를 지정해주고 SIGINT시그널이 들어오는 경우 int_handler가 처리하도록 하였습니다. int_handler는 자식에게 kill함수를 사용하여 인터럽트 시그널을 보내고 wait함수를 사용하여 대기합니다. 그후 자식프로세스를 종료하였다는 문구를 출력후 exit함수로 자신(부모)를 종료합니다.

자식 프로세스

1. 부모로부터 주문(SIGUSR1)을 받으면 조리를 시작
- main함수에서 sigusr1함수가 들어오면 sigaction함수를 통해 sigusr1핸들러가 처리하도록 동작합니다.(조리시작)

2. 일단 조리를 시작하면 음식이 완성되기 전까지 SIGALARM을 제외한 다른 시그널은 모두 무시
- sigaction 함수를 사용하여 sa_handler를 SIG_IGN으로 설정 후 for루프문을 사용하여 SIGALRM시그널을 제외한 다른시그널들을 무시하도록 설정해주었습니다.

3. 일정 조리시간이 지내면(alarm 이용) 음식이 완성되고, 이를 부모 프로세스에게 알린다(SIGUSR1)
sigaction으로 alrm시그널이 올 경우 alrm핸들러에서 처리하게되고 완성된 요리를 전달한다라는 문구를 출력합니다. 그후 kill함수를 사용하여 부모프로세스에 SIGUSR1시그널을 보냅니다. (kill(getppid(),SIGUSR1))

4. 음식이 완성되면 다른 시그널을 받을 수 있도록 전환
- sigaction 함수를 사용하여 sa_handler를 SIG_DFL로 설정후 for루프문을 사용하여 시그널들을 디폴트 동작으로 수행되도록 바꾸어주었습니다.

확인표 작성

확인사항	완성여부	비고
1. (부모) 자식 프로세스 생성 및 exec 정상 작동 여부	O	
2. (부모) 메뉴 출력 및 사용자로부터 입력 받기	O	
3. (부모) 주문 받기 전 Ctrl-C 받았을 때 정상 동작 (메시지 출력 / 자식 프로세스 종료 / 자식 종료) 여부	O	
4. (부모) 주문 받은 후 관련 시그널 처리 준비한 후 자식에게 SIGUSR1 보내기	O	
5, (부모) 조리가 시작된 후 다른 시그널 무시	O	
6. (자식) 시그널 처리 준비	O	
7. (자식) SIGUSR1 받았을 때 관련 시그널 처리 준비한 후 조리 시작 (화면에 메시지 출력)	O	
8. (자식) SIGALARM 받았을 때 관련 시그널 처리 준비한 후 부모에게 통보	O	
9. (부모) 자식으로부터 SIGUSR1 받았을 때 관련 시그널 처리 준비한 후 손님에게 메시지 출력	O	
10. (부모) 음식이 완료된 후 Ctrl-C 받았을 때 정상 동작 (메시지 출력 / 자식 프로세스 종료 / 자식 종료) 여부	O	

출력결과

makefile

```
[khm970514@lily ch7]$ ls
\         c_cook_bak.c   ex7_1.c    ex7_10.exe  ex7_12.c    ex7_13.exe  ex7_3.c    ex7_5.c    ex7_6.exe  ex7_8.c    ex7_9.exe     p_cook_bak2.c
Makefile  c_cook_bak2.c  ex7_1.exe  ex7_11.c    ex7_12.exe  ex7_2.c     ex7_3.exe  ex7_5.exe  ex7_7.c    ex7_8.exe  p_cook.c      p_cook_bak3.c
c_cook.c  c_ook.exe      ex7_10.c   ex7_11.exe  ex7_13.c    ex7_2.exe   ex7_4.c    ex7_6.c    ex7_7.exe  ex7_9.c    p_cook_bak.c  p_cook_bak4.c
[khm970514@lily ch7]$ make
make p_cook.exe
make[1]: Entering directory `/home/2019/UNIX/khm970514/unix/ch7'
gcc -o p_cook.exe p_cook.c
make[1]: Leaving directory `/home/2019/UNIX/khm970514/unix/ch7'
make c_cook.exe
make[1]: Entering directory `/home/2019/UNIX/khm970514/unix/ch7'
gcc -o c_cook.exe c_cook.c
make[1]: Leaving directory `/home/2019/UNIX/khm970514/unix/ch7'
[khm970514@lily ch7]$ ls
\         c_cook.exe     c_ook.exe  ex7_10.c    ex7_11.exe  ex7_13.c    ex7_2.exe  ex7_4.c    ex7_6.c    ex7_7.exe  ex7_9.c    p_cook.exe     p_cook_bak3.c
Makefile  c_cook_bak.c   ex7_1.c    ex7_10.exe  ex7_12.c    ex7_13.exe  ex7_3.c    ex7_5.c    ex7_6.exe  ex7_8.c    ex7_9.exe  p_cook_bak.c   p_cook_bak4.c
c_cook.c  c_cook_bak2.c  ex7_1.exe  ex7_11.c    ex7_12.exe  ex7_2.c     ex7_3.exe  ex7_5.exe  ex7_7.c    ex7_8.exe  p_cook.c   p_cook_bak2.c
[khm970514@lily ch7]$
```

확인표

1, 2, 4, 6, 7, 8, 9 (정상적인 루틴)

```
[khm970514@lily ch7]$ ./p_cook.exe
---------- parent process ----------
음식을 주문하시겠습니까? (y/n) : y
주문을 요청하였습니다.
---------- child process ----------
주문이 접수되었습니다.
완성된 요리를 전달합니다.
---------- parent process ----------
완성된 요리를 전달받아 손님에게 제공하였습니다.
---------- parent process ----------
음식을 주문하시겠습니까? (y/n) : y
주문을 요청하였습니다.
---------- child process ----------
주문이 접수되었습니다.
완성된 요리를 전달합니다.
---------- parent process ----------
완성된 요리를 전달받아 손님에게 제공하였습니다.
---------- parent process ----------
음식을 주문하시겠습니까? (y/n) : n
주문프로그램을 종료합니다.
[khm970514@lily ch7]$
```

3. 주문 받기전 인터럽트 동작

```
[khm970514@lily ch7]$ ./p_cook.exe
---------- parent process ----------
음식을 주문하시겠습니까? (y/n) : ^C
자식 프로세스를 종료하였습니다.
[khm970514@lily ch7]$
```

5. 주문을 받은후 조리가 완료될때까지는 시그널 무시

```
[khm970514@lily ch7]$ ./p_cook.exe
---------- parent process ----------
음식을 주문하시겠습니까? (y/n) : y
주문을 요청하였습니다.
---------- child process ----------
주문이 접수되었습니다.
^C^C^C^C^C^C^C^C^C^C^C^C^C^C^C^C^C^C^C^C^C^C^C^C^C^C^C완성된 요리를 전달합니다.
---------- parent process ----------
완성된 요리를 전달받아 손님에게 제공하였습니다.
---------- parent process ----------
음식을 주문하시겠습니까? (y/n) : n
주문프로그램을 종료합니다.
[khm970514@lily ch7]$
```

10. 음식 완료후에 인터럽트 발생하는 경우

```
[khm970514@lily ch7]$ ./p_cook.exe
---------- parent process ----------
음식을 주문하시겠습니까? (y/n) : y
주문을 요청하였습니다.
---------- child process ----------
주문이 접수되었습니다.
완성된 요리를 전달합니다.
---------- parent process ----------
완성된 요리를 전달받아 손님에게 제공하였습니다.
---------- parent process ----------
음식을 주문하시겠습니까? (y/n) : y
주문을 요청하였습니다.
---------- child process ----------
주문이 접수되었습니다.
완성된 요리를 전달합니다.
---------- parent process ----------
완성된 요리를 전달받아 손님에게 제공하였습니다.
---------- parent process ----------
음식을 주문하시겠습니까? (y/n) : ^C
자식 프로세스를 종료하였습니다.
[khm970514@lily ch7]$
```