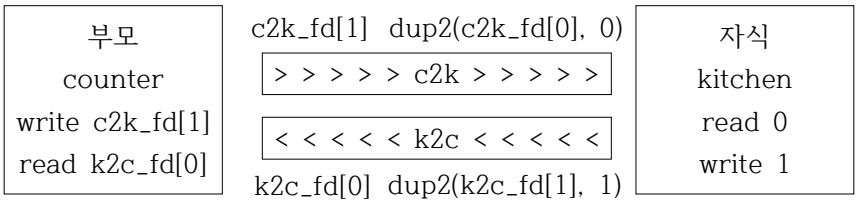
응용12

pipe 설계



1. 부모는 자식에게 문자열로 메뉴입력받아 파이프로 전달

 - pipe를 사용하여 파이프를 생성, 부모자식 프로세스간 통신에 사용하였습니다. 통신에 사용된 파이프, 파일디스크립터는 위와 같습니다.

2. 자식은 파이프로 문자열을 전달받아 fprintf(stderr)로 출력, 부모에게 전달할 문자열을 파이프에 작성

 - 부모에서 fgets함수를 사용하여 stdin으로 메뉴를 입력받은 후 write함수를 호출하여 c2k_fd[1]에 쓰기해주어 파이프에 기록하였습니다. 자식은 부모가 기록한 파이프를 exec이전 dup2함수를 사용하여 0과 1로 사용하기로 하였기 때문에 read(0)으로 파이프 내용을 읽어오고 write(1)로 파이프에 기록합니다. 자식은 알람이 호출되면 success라는 문자열을 파이프에 기록하고 부모는 k2c_fd[0]을 읽어 success라는 문자와 일치하면 음식을 제공합니다.

3. alrm함수를 사용하여 요리시작

 - alrm함수를 사용하여 알람시간을 매개변수로 넣고 시그널을 발생시킵니다. 이때의 alrm시그널은 미리 작성해둔 알람 핸들러가 처리하게됩니다. (응용 10과 동일)

4. 부모가 자식이 파이프에 쓰기한 것을 확인 후 자식이 파이프에 작성한 문자열을 불러와 비교

 - 자식이 write(1)을 통해 파이프에 success라는 문자열을 기록하면 부모는 read를 사용하여 k2c_fd[0]에 무엇인가 기록되면 success라는 문자열과 비교하여 문자열이 동일하면 음식을 제공한다는 문구를 출력합니다.

6. 반복 수행

 - 반복적으로 수행하도록 메뉴 입력부터 완성 종료까지를 while(1)로 무한루프 돌려주었고, 자식또한 종료되지 않도록 while(1)로 무한루프를 돌린 후 파이프에 무엇인가 기록될 경우 동작하도록 하였습니다.

Makefile

```
# Makefile

cc = gcc
CFLAG = -g

all :
	make counter.exe
	make kitchen.exe

counter.exe : counter.c
	gcc -o counter.exe counter.c

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

counter.c

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

#define MENU_SIZE 128

int pid_child = -1;

void int_handler(int signo)
{
        kill(pid_child, SIGINT); //자식종료
        printf("\n\n[부모] 자식 프로세스를 종료하였습니다.\n");
        printf("[부모] 자신을 종료합니다.\n");
        exit(0);
}

int main(void)
{
        int c2k_fd[2];
        int k2c_fd[2];

        if(pipe(c2k_fd) == -1)
        {
                perror("pipe");
                exit(1);
        }
        if(pipe(k2c_fd) == -1)
        {
                perror("pipe");
                exit(1);
        }

        pid_child = fork();

        if(pid_child < 0) //fork 오류
        {
                perror("fork");
                exit(2);
        }
        else if(pid_child == 0) //자식프로세스
        {
                close(c2k_fd[1]);
                close(k2c_fd[0]);

                dup2(c2k_fd[0], 0); // stdin -> c2k_fd[0]
                dup2(k2c_fd[1], 1); // stdout -> k2c_fd[1]

                if(execl("./kitchen.exe", "kitchen.exe", NULL) < 0)
                {
                        perror("execl");
                        exit(2);
                }
                close(c2k_fd[0]);
                close(k2c_fd[1]);
        }
```

```
	else
	{
		close(c2k_fd[0]);
		close(k2c_fd[1]);

		char menu[MENU_SIZE];
		while(1)
		{
			struct sigaction act; //메뉴 동작 전 인터럽트 발생시 핸들러처리하도록
			sigemptyset(&act.sa_mask);
			act.sa_flags = 0;
			act.sa_handler = int_handler;
			if(sigaction(SIGINT, &act, (struct sigaction *)NULL) < 0)
			{
				perror("sigaction 에러 발생");
				exit(3);
			}

			printf("[부모] 주문하실 메뉴를 입력해주세요 (문자열) : ");
			fgets(menu, MENU_SIZE, stdin);
			if(write(c2k_fd[1], menu, strlen(menu)) != -1) //파이프에 쓰기성공시
			{
				printf("[부모] 메뉴를 전달하였습니다.\n\n");
			}
			else
			{
				perror("write 에러 발생");
				exit(4);
			}

			act.sa_handler = SIG_IGN; //요리 완료전 SIGINT, SIGQUIT 무시
			if(sigaction(SIGINT, &act, (struct sigaction *)NULL) < 0)
			{
				perror("sigaction 에러 발생");
				exit(3);
			}
			if(sigaction(SIGQUIT, &act, (struct sigaction *)NULL) < 0)
			{
				perror("sigaction 에러 발생");
				exit(3);
			}

			//after child process

			char buf[BUFSIZ];
			if(read(k2c_fd[0], buf, BUFSIZ) != -1) //자식에게 파이프로 완료를 전달받았을때
			{
				if(strncmp("success", buf, 7) == 0)
				{
					printf("[부모] 메뉴를 전달받아 손님에게 제공하였습니다.\n");
				}
			}
			else
			{
				perror("read 에러 발생");
				exit(5);
			}
		}
		close(c2k_fd[1]);
		close(k2c_fd[0]);
	}
	return 0;
}
```

<소스코드>

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

#define MENU_SIZE 128

int pid_child = -1;

void int_handler(int signo)
{
        kill(pid_child, SIGINT); //자식종료
        printf("₩n₩n[부모] 자식 프로세스를 종료하였습니다.₩n");
        printf("[부모] 자신을 종료합니다.₩n");
        exit(0);
}

int main(void)
{
        int c2k_fd[2];
        int k2c_fd[2];

        if(pipe(c2k_fd) == -1)
        {
                perror("pipe");
                exit(1);
        }
        if(pipe(k2c_fd) == -1)
        {
                perror("pipe");
                exit(1);
        }

        pid_child = fork();

        if(pid_child < 0) //fork 오류
        {
                perror("fork");
                exit(2);
        }
        else if(pid_child == 0) //자식프로세스
        {
                close(c2k_fd[1]);
                close(k2c_fd[0]);

                dup2(c2k_fd[0], 0); // stdin -> c2k_fd[0]
                dup2(k2c_fd[1], 1); // stdout -> k2c_fd[1]

                if(execl("./kitchen.exe", "kitchen.exe", NULL) < 0)
                {
                        perror("execl");
                        exit(2);
                }
                close(c2k_fd[0]);
                close(k2c_fd[1]);
        }
        else
```

```
	{
		close(c2k_fd[0]);
		close(k2c_fd[1]);

		char menu[MENU_SIZE];
		while(1)
		{
			struct sigaction act; //메뉴 동작 전 인터럽트 발생시 핸들러처리하도록
			sigemptyset(&act.sa_mask);
			act.sa_flags = 0;
			act.sa_handler = int_handler;
			if(sigaction(SIGINT, &act, (struct sigaction *)NULL) < 0)
			{
				perror("sigaction 에러 발생");
				exit(3);
			}

			printf("[부모] 주문하실 메뉴를 입력해주세요 (문자열)  : ");
			fgets(menu, MENU_SIZE, stdin);
			if(write(c2k_fd[1], menu, strlen(menu)) != -1) //파이프에 쓰기성공시
			{
				printf("[부모] 메뉴를 전달하였습니다.₩n₩n");
			}
			else
			{
				perror("write 에러 발생");
				exit(4);
			}
			act.sa_handler = SIG_IGN; //요리 완료전 SIGINT, SIGQUIT 무시
			if(sigaction(SIGINT, &act, (struct sigaction *)NULL) < 0)
			{
				perror("sigaction 에러 발생");
				exit(3);
			}
			if(sigaction(SIGQUIT, &act, (struct sigaction *)NULL) < 0)
			{
				perror("sigaction 에러 발생");
				exit(3);
			}
			//after child process

			char buf[BUFSIZ];
			if(read(k2c_fd[0], buf, BUFSIZ) != -1) //자식에게 파이프로 완료를 전달받았을때
			{
				if(strncmp("success", buf, 7) == 0)
				{
					printf("[부모] 메뉴를 전달받아 손님에게 제공하였습니다.₩n");
				}
			}
			else
			{
				perror("read 에러 발생");
				exit(5);
			}
		}
		close(c2k_fd[1]);
		close(k2c_fd[0]);
	}
	return 0;
}
```

kitchen.c

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

#define TIME_FOR_COOK 5

void alrm_handler(int signo)
{
        fprintf(stderr, "[자식] 완성된 요리를 전달합니다.\n\n");
        if(write(1, "success", 7) == -1) //알람시그널이 호출되면 파이프에 success기록
        {
                perror("write 에러 발생");
                exit(2);
        }

        struct sigaction act; //무시했던 시그널 디폴트로 돌려줌
        sigemptyset(&act.sa_mask);
        act.sa_flags = 0;
        act.sa_handler = SIG_DFL;
        if(sigaction(SIGINT, &act, (struct sigaction *)NULL) < 0)
        {
                perror("sigaction 에러발생");
                exit(4);
        }
        if(sigaction(SIGQUIT, &act, (struct sigaction *)NULL) < 0)
        {
                perror("sigaction 에러발생");
                exit(4);
        }
}

int main(void)
{
        char buf[BUFSIZ];
        while(1)
        {
                if(read(0, buf, BUFSIZ) != -1) //부모가 파이프로 전송하는경우 동작
                {
                        fprintf(stderr,"[자식] 주문을 전달받았습니다. 메뉴 : %s", buf);

                        struct sigaction act;
                        sigemptyset(&act.sa_mask);
                        act.sa_flags = 0;
                        act.sa_handler = SIG_IGN;
                        if(sigaction(SIGINT, &act, (struct sigaction *)NULL) < 0)
                        {
                                perror("sigaction 에러발생");
                                exit(4);
                        }
                        if(sigaction(SIGQUIT, &act, (struct sigaction *)NULL) < 0)
                        {
                                perror("sigaction 에러발생");
                                exit(4);
                        }

                        //알람 시그널 발생시 처리할 핸들러 지정
                        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(4);
                        }
                        fprintf(stderr,"[자식] 조리를 시작합니다.\n");

                        alarm(TIME_FOR_COOK); //알람발생(핸들러 호출)
                }
        }
        return 0;
}
```

<소스코드>

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

#define TIME_FOR_COOK 5

void alrm_handler(int signo)
{
	fprintf(stderr, "[자식] 완성된 요리를 전달합니다.\n\n");
	if(write(1, "success", 7) == -1) //알람시그널이 호출되면 파이프에 success기록
	{
		perror("write 에러 발생");
		exit(2);
	}

	struct sigaction act; //무시했던 시그널 디폴트로 돌려줌
	sigemptyset(&act.sa_mask);
	act.sa_flags = 0;
	act.sa_handler = SIG_DFL;
	if(sigaction(SIGINT, &act, (struct sigaction *)NULL) < 0)
	{
		perror("sigaction 에러발생");
		exit(4);
	}
	if(sigaction(SIGQUIT, &act, (struct sigaction *)NULL) < 0)
	{
		perror("sigaction 에러발생");
		exit(4);
	}
}

int main(void)
{
	char buf[BUFSIZ];
	while(1)
	{
		if(read(0, buf, BUFSIZ) != -1) //부모가 파이프로 전송하는경우 동작
		{
			fprintf(stderr,"[자식] 주문을 전달받았습니다. 메뉴 : %s", buf);

			struct sigaction act;
			sigemptyset(&act.sa_mask);
			act.sa_flags = 0;
			act.sa_handler = SIG_IGN;
			if(sigaction(SIGINT, &act, (struct sigaction *)NULL) < 0)
			{
				perror("sigaction 에러발생");
				exit(4);
			}
			if(sigaction(SIGQUIT, &act, (struct sigaction *)NULL) < 0)
			{
				perror("sigaction 에러발생");
				exit(4);
```

```
                        }

                        //알람 시그널 발생시 처리할 핸들러 지정
                        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(4);
                        }
                        fprintf(stderr,"[자식] 조리를 시작합니다.\n");

                        alarm(TIME_FOR_COOK); //알람발생(핸들러 호출)
                }
        }
        return 0;
}
```

체크리스트

확인사항	완성여부	비고
1. (부모) 파이프 생성 후 자식 프로세스 생성 및 작동 여부	O	
2. (부모) 파이프 준비 / 메뉴 출력 및 사용자로부터 입력 받기	O	
3. (부모) 주문 받기 전 Ctrl-C 받았을 때 정상 동작 (메시지 출력 / 자식프로세스 종료 / 자신 종료) 여부	O	
4. (부모) 주문 받은 후 자식에게 주문 보내기	O	
5. (부모) 요리가 완료되지 않은 경우 SIGINT, SIGQUIT 시그널 무시	O	
6. (자식) 시그널 처리 준비	O	
7. (자식) 주문을 받았을 때 조리 시작 (화면에 메시지 출력	O	
8. (자식) SIGALARM 받았을 때 부모에게 통보	O	
9. (부모) 자식으로부터 통보를 받았을 때 손님에게 메시지 출력	O	
10. (부모) 모든 음식이 완료된 후 Ctrl-C 받았을 때 정상 동작 (메시지 출력 / 자식 프로세스 종료 / 자신 종료) 여부	O	

make

```
[khm970514@lily ch7]$ make
make counter.exe
make[1]: Entering directory `/home/2019/UNIX/khm970514/unix/ch7'
gcc -o counter.exe counter.c
make[1]: Leaving directory `/home/2019/UNIX/khm970514/unix/ch7'
make kitchen.exe
make[1]: Entering directory `/home/2019/UNIX/khm970514/unix/ch7'
gcc -o kitchen.exe kitchen.c
make[1]: Leaving directory `/home/2019/UNIX/khm970514/unix/ch7'
[khm970514@lily ch7]$
```

실행결과

1. (부모) 파이프 생성 후 자식 프로세스 생성 및 작동 여부
2. (부모) 파이프 준비 / 메뉴 출력 및 사용자로부터 입력 받기
4. (부모) 주문 받은 후 자식에게 주문 보내기

```
[khm970514@lily ch7]$ ./counter.exe
[부모] 주문하실 메뉴를 입력해주세요 (문자열) : 김밥
[부모] 메뉴를 전달하였습니다.

[자식] 주문을 전달받았습니다. 메뉴 : 김밥
[자식] 조리를 시작합니다.
[자식] 완성된 요리를 전달합니다.

[부모] 메뉴를 전달받아 손님에게 제공하였습니다.
[부모] 주문하실 메뉴를 입력해주세요 (문자열) : 치킨
[부모] 메뉴를 전달하였습니다.

[자식] 주문을 전달받았습니다. 메뉴 : 치킨
[자식] 조리를 시작합니다.
[자식] 완성된 요리를 전달합니다.

[부모] 메뉴를 전달받아 손님에게 제공하였습니다.
[부모] 주문하실 메뉴를 입력해주세요 (문자열) : 우동
[부모] 메뉴를 전달하였습니다.

[자식] 주문을 전달받았습니다. 메뉴 : 우동
[자식] 조리를 시작합니다.
^C^C^C^\^\^\^\[자식] 완성된 요리를 전달합니다.

[부모] 메뉴를 전달받아 손님에게 제공하였습니다.
[부모] 주문하실 메뉴를 입력해주세요 (문자열) : ^C

[부모] 자식 프로세스를 종료하였습니다
[부모] 자신을 종료합니다.
[khm970514@lily ch7]$
```

6. (자식) 시그널 처리 준비
7. (자식) 주문을 받았을 때 조리 시작
(화면에 메시지 출력)

8. (자식) SIGALARM 받았을 때
부모에게 통보
9. (부모) 자식으로부터 통보를
받았을 때 손님에게 메시지 출력

5. (부모) 요리가 완료되지 않은 경우
SIGINT, SIGQUIT 시그널 무시

10. (부모) 모든 음식이 완료된 후 Ctrl-C 받았을 때 정상 동작
(메시지 출력 /자식 프로세스 종료 / 자신 종료) 여부

```
[khm970514@lily ch7]$ ./counter.exe
[부모] 주문하실 메뉴를 입력해주세요 (문자열) : ^C

[부모] 자식 프로세스를 종료하였습니다.
[부모] 자신을 종료합니다.
[khm970514@lily ch7]$
```

3. (부모) 주문 받기 전 Ctrl-C 받았을 때 정상 동작
(메시지 출력 / 자식프로세스 종료 / 자신 종료) 여부