**1) 프로그램 코드 및 설명**

**ㄱ) fork() 와 pipe() 를 이용한 Producer & Consumer 구현 :**

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

#define BUFFERSIZE 256 // 버퍼 사이즈 지정

int main(void)

{

char buffer[BUFFERSIZE];

int fd[2];

int i;

if(pipe(fd) == -1) // 파이프 생성

{

perror("pipe error.. \n");

exit(0);

}

pid\_t pid;

pid = fork();// fork 호출

if(pid == -1)

{

perror("fork error... \n");

exit(0);

}

else if(pid == 0) // 자식 프로세스 (Producer)

{

for(i = 0; i < 10 ; i++)

{

printf("Produce an item : Item\n");

write(fd[1], “Item”, BUFFERSIZE); //pipe를 통해 write한다.

}

}

else { // 부모 프로세스 (Consumer)

for(i = 0; i<10 ; i++)

{

int nread;

while((nread = read(fd[0], buffer, BUFFERSIZE))<BUFFERSIZE); // Producer가 생산할때까지 대기

printf("Consume the item : %s \n", buffer); // 출력 (소비)

}

}

exit(0);

}

- 파이프를 생성하고, fork()를 호출하여 자식 프로세스를 생성한다. 그리고 자식 프로세스(Producer)는 파이프를 통하여 write한다. 즉, item을 생성한다. 부모 프로세스(Consumer)는 자식 프로세스(Producer)가 생산 할 때까지 while문으로 read를 하면서 대기하고, 생산이 되면 read하여서 소비한다. 즉, 생산이 되지 않았는데 Consumer가 소비하는 경우를 방지한다.

**ㄴ) fork() 와 pipe() 를 이용한 Producer & Consumer 구현 2 :**

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

#define BUFFERSIZE 64 //버퍼 사이즈 지정

int main(void)

{

char buffer[BUFFERSIZE]; // Consumer에서 Producer로부터 pipe를 통해 생산받은 데이터를 받을 버퍼

char bufcheck[BUFFERSIZE]; //버퍼가 다찼지 확인하는 버퍼

int fd[2]; // 생산, 소비를 위한 파이프 (이하 파이프1)

int fd2[2]; // 버퍼가 다 찼는지 통신하기위한 파이프 (이하 파이프2)

int i;

int in,out=0; //버퍼의 참조 인덱스값

if(pipe(fd) == -1) //파이프1 생성

{

perror("pipe error.. \n");

exit(0);

}

if(pipe(fd2) == -1)

{

perror("pipe error.. \n");

exit(0);

}

pid\_t pid;

pid = fork();

if(pid == -1) // 파이프2 생성

{

perror("fork error... \n");

exit(0);

}

else if(pid == 0)//자식프로세스(Producer)

{

int nread=0;

for(i = 0; i < 200 ; i++) // 200번 반복 (버퍼사이즈는 64)

{

printf("Produce an item\n");

in++;

in %= BUFFERSIZE;

write(fd[1], "A", 2); //파이프1을 통해 2바이트씩 생산

if((nread = read(fd2[0], bufcheck ,BUFFERSIZE))-1 == (in+1)%BUFFERSIZE)

//파이프2를 통하여 버퍼가 가득찼는지 확인

{

printf("Buffer is Full\n");

sleep(1); //가득찼을 시 Consumer가 소비할때까지 대기

}

}

}

else //부모 프로세스(Consumer)

{

int nread;

for(i = 0; i<200 ; i++)

{

write(fd2[1],"Buffer is Full Signal",out+1); //파이프2를 통해 Consumer에서 참조하는 인덱스값을 (버퍼가 가득찼는지를 알리기위해) Producer에게 전송

while((nread = read(fd[0], buffer, 2)) <= 0); //Producer가 생산을 하면 읽어와 Consume한다. (생산이 안되있는데 소비하는 경우를 방지)

printf("Consume the item : %s / Buffer : %d\n", buffer, out);

out++;

out %= BUFFERSIZE;

}

}

exit(0);

}

- ㄱ)의 프로그램과는 다르게 파이프를 하나의 배열로 가정하고(공유 변수를 사용할 수 없고, 배열(void\*)로만 프로세스간 서로 통신을 하는 파이프의 제한 때문에), fork()와 pipe()를 이용하여 구현하였다. in과 out은 Producer(자식 프로세스)와 Consumer(부모 프로세스)가 그 배열을 참조하는 index이며, 파이프는 2개를 생성해, 1개는 Producer에서 생산하여 Consumer에서 소비시키기 위한 파이프이며, 나머지 1개는 버퍼가 가득 찼는지 Consumer에서 Producer로 알리기 위한 파이프이다. 버퍼를 끝까지 참조했을 경우 (버퍼가 가득 차면) 버퍼를 참조하는 index값은 0으로 돌아가고(버퍼를 비우고) 다시 버퍼의 처음 인덱스(0)부터 생산을 시작한다. Consumer도 Producer의 생산과 마찬가지로 소비를 한다. while문을 통하여 Producer가 생산을 하지 않았는데 Consumer가 소비하는 경우를 방지하고, if문을 통하여 버퍼가 가득 찼는지 확인하고, 가득 찼을 경우 Consumer가 소비소비 할 때까지 Producer는 1초간 대기한다.

**ㄷ) p\_thread() 를 이용한 Producer & Consumer 구현 :**

#include <stdio.h>

#include <pthread.h>

#include <stdlib.h>

void \*consumer(void \*pData); //Consumer

char buffer[10];

int n = 10;

int in = 0;

int out = 0;

int main(int argc, char \*\*argv)

{

char nextp; int i;

pthread\_t tid;

if( pthread\_create(&tid,NULL,consumer,NULL) != 0) // 쓰레드 생성

{

printf("fail to pthread\_create \n");

return 0;

}

for(i = 0; i < 50; i++)

{

nextp = 'P';

while((in+1)%n == out); //버퍼가 꽉차면 Consumer가 소비할때까지 대기

printf("Produce an item : %c\n", nextp);

buffer[in] = nextp; // 공유하는 버퍼에 넣는다.

in++;

in %= n; //버퍼를 참조하는 인덱스 값 설정(인덱스 값이n 까지 갈경우 인덱스 0부터 다시 참조)

}

pthread\_join(tid,NULL);

}

void \*consumer(void \*pData)

{

int i;

char nextc;

for(i=0; i<50; i++)

{

while(in == out); // Producer가 생산할때까지 대기

nextc = buffer[out];

out++;

out %= n; //버퍼를 참조하는 인덱스 값 설정(인덱스 값이 n까지갈경우 인덱스 0부터다시 참조)

printf("Consume the item : %c \n",nextc);

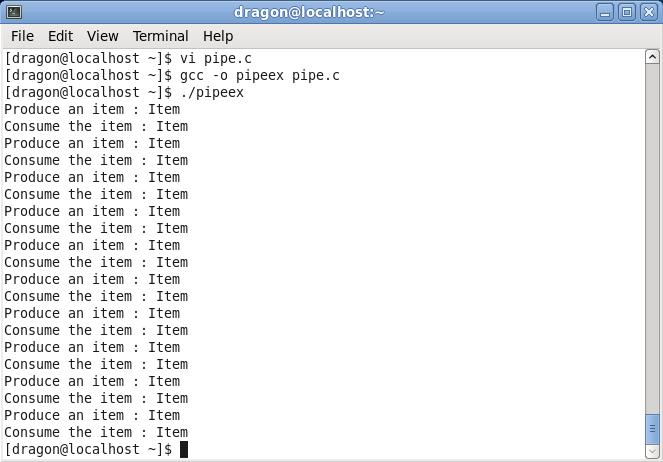
}

}

- p\_thread()를 호출하여 쓰레드(Consumer)를 생성한다. 메인과 쓰레드가 병행으로 실행되며, 배열을 참조하는 index값과 while문을 이용한 대기를 통해, Producer가 생산하지 않았는데 Consumer가 소비하는 경우와 버퍼가 가득 차있는데, 생산을 계속 하는 경우를 방지 한다. 버퍼를 끝까지 참조 했을 경우, 배열을 참조하는 index값은 0으로 돌아가고(버퍼를 비우고) 다시 버퍼의 처음 인덱스(0)부터 생산을 시작한다. Consumer도 이와 마찬가지로, 버퍼를 끝까지 참조했을 경우, 다시 버퍼의 처음 인덱스(0)부터 소비를 시작한다.

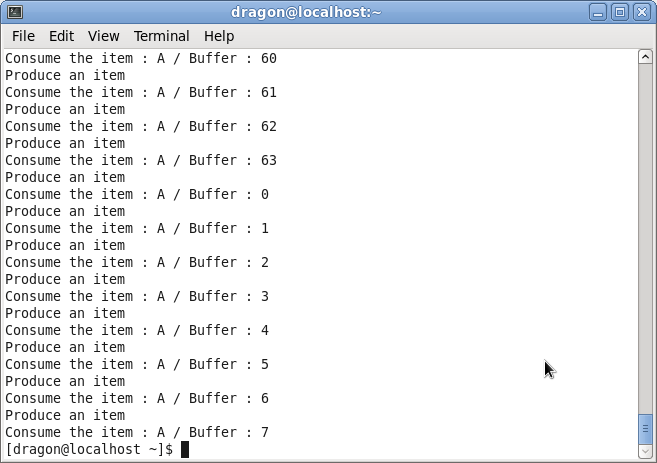
**2) 프로그램 실행 화면 :**

**ㄱ)**



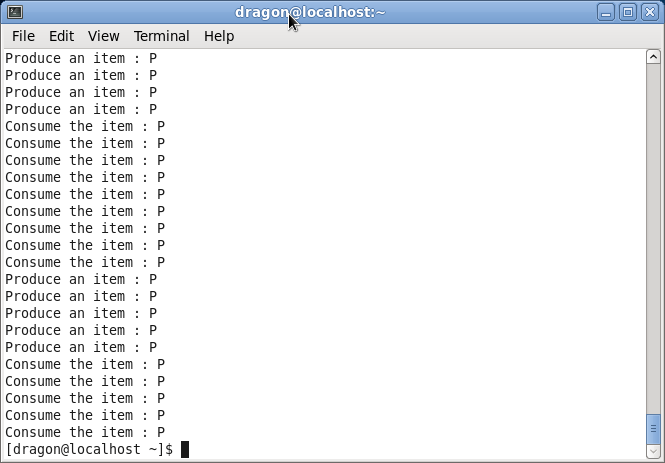
- fork() 와 pipe() 를 이용한 Producer & Consumer 구현 출력 결과.

**ㄴ)**



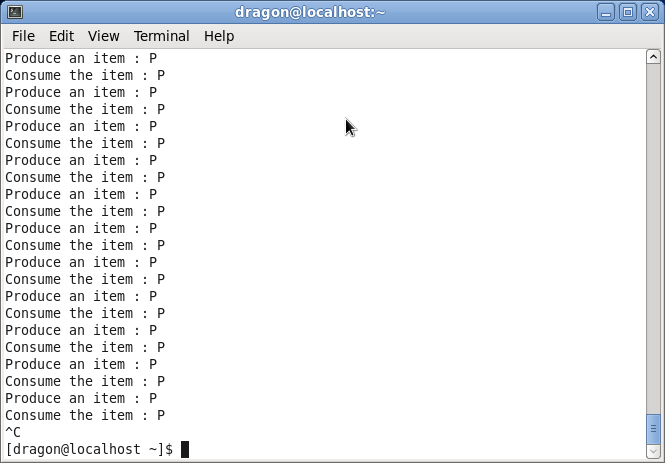
- fork() 와 pipe() 를 이용한 Producer & Consumer 구현2 출력 결과. Buffer는 현재 참조하는 배열(파이프)의 인덱스 값이다.

**ㄷ)**



- p\_thread() 를 이용한 Producer & Consumer 구현 출력 결과. Producer가 생산한만큼 Consumer가 소비한다.

**ㄷ2)**



-p\_thread() 를 이용한 Producer & Consumer 구현 출력 결과. 이 결과는 Producer가 생산을 할 때 sleep()을 1초 만큼 걸어서 실행을 시켰을 경우이다.