**201611229 장정아 운영체제 Producer-Consumer 문제**

**1. fork 로 child 프로세스를 생성하여 운영하는 방식**

**1.1 코드**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

#include <fcntl.h>

#include <ctype.h>

#include <string.h>

#include <errno.h>

#include <sys/wait.h>

void producer(FILE \*pipe\_write\_end)

{

int i;

for (i = 1; i <= 100; i++) {

fprintf(pipe\_write\_end, "%d ", i); //item 생성

printf("produce an item %d\n", i);

}

fclose(pipe\_write\_end);

exit(0);

}

void consumer(FILE \*pipe\_read\_end)

{

int n, k;

while (1) {

int n = fscanf(pipe\_read\_end, "%d", &k);

//pipe에서 item 읽어오기

if (n == 1) printf("consume the item %d\n", k);

else break;

}

fclose(pipe\_read\_end);

exit(0);

}

int main()

{

pid\_t producer\_id, consumer\_id;

int filedes[2];

FILE \*pipe\_write\_end, \*pipe\_read\_end;

// 쓰기전용, 읽기전용 파이프 생성

pipe(pd);

pipe\_read\_end = fdopen(filedes[0], "r");

pipe\_write\_end = fdopen(filedes[1], "w");

// 읽기 권한, 쓰기 권한으로 스트림 열기

p\_id = fork();

if (p\_id == 0) {

fclose(pipe\_read\_end);

producer(pipe\_write\_end);

} //producer 프로세스 생성

c\_id = fork();

if (c\_id == 0) {

fclose(pipe\_write\_end);

consumer(pipe\_read\_end);

} //consumer 프로세스 생성

fclose(pipe\_read\_end);

fclose(pipe\_write\_end);

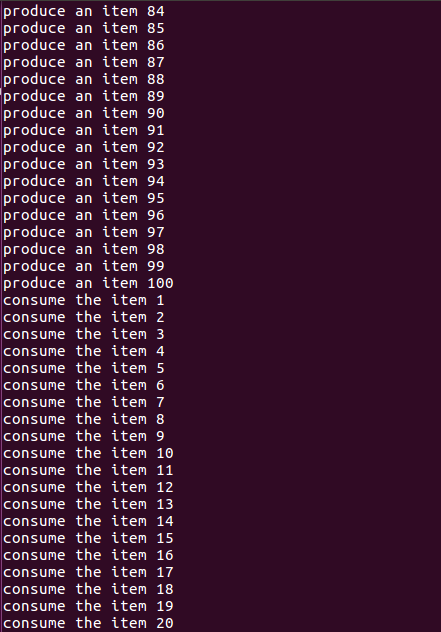
wait(NULL);

wait(NULL); // producer 과 consumer 프로세스 대기

return 0;

}

**1.2 실행 결과**



**2. pthread 로 쓰레드를 생성하여 운영하는 방식**

**2.1 코드**

#include <stdio.h>

#include <pthread.h>

#include <stdlib.h>

#define NUM\_THREAD 2

int buff[100];

int in = -1; int out = -1; // 각각 producer, consumer 의 인덱스를 의미

void\* consumer(void\* arg);

void\* producer(void\* arg);

int main()

{

int num = 0;

int rc = 0, i = 0;

pthread\_t tid[NUM\_THREAD];

int result[NUM\_THREAD];

pthread\_create(&tid[0], NULL, consumer, (void \*)&num);

pthread\_create(&tid[1], NULL, producer, (void \*)&num);

//producer, consumer thread 생성

for (i = 0; i<NUM\_THREAD; i++){

rc = pthread\_join(tid[i], (void\*\*)&result[i]);

if (rc != 0){

printf("Error in thread[%d] : %d\n", i, rc);

exit(i);

}

} // thread가 수행을 마치면 join

return 0;

}

void\* consumer(void\* arg){

int i; int next;

for (i = 0; i < 500; i++) {

while (in == out); // producer가 buff에 물건을 채워넣기를 기다림

next = buff[out]; // next 는 사용자가 사용할 물건

out++; out %= 100; // 100개 단위로 사용하기로 임의로 설정

printf("consume the item %d\n", next);

}

}

void\* producer(void\* arg){

int i;

for (i = 0; i < 500; i++){

printf("produce an item %d\n", i);

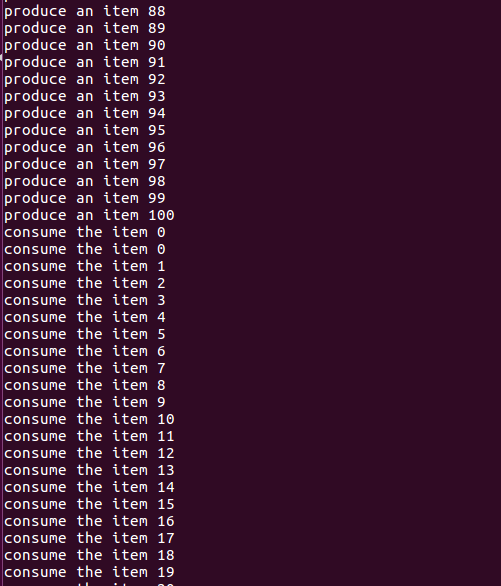
while ((in + 1) % 100 == out); // consumer가 buff에서 물건을 사용하기를 기다림

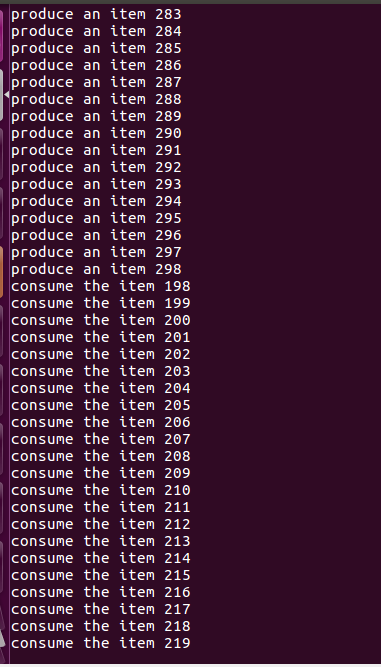
in++; in %= 100; buff[in] = i; // 생산한 물건을 버퍼에 넣음 (100개 단위로 넣기로 임의로 설정)

} }

**2.2 실행 결과**

* Producer 와 Consumer 스레드가 서로 번갈아가면서 실행





**3. 요점 정리**

**3.1 fork 로 child 프로세스를 생성하여 운영하는 방식**

- 프로세스로 운영을 하려면 우선 프로세스간 구분을 할 수 있어야한다. 이때, fork() 시 부모 프로세스는 pid값을 갖고 막 생성된 자식 프로세스( producer, consumer )들은 pid값이 0이기 때문에 pid값으로 구분해주면 된다.

- 또한 프로세스는 메모리가 독립적으로 존재하기 때문에 fork()로 생성된 프로세스들은 프로세스끼리 데이터를 주고 받을 방법이 없는데, 이 때 프로세스간 정보교환을 원활히 하려면 pipe() 라는 함수를 사용하면 된다. Pipe()는 하나의 파이프 및 파이프에 대한 2개의 파일 디스크립터 ( 읽기, 쓰기 ) 를 생성한다. 이 파이프를 두 프로세스가 공유한다. Producer-consumer 문제를 풀 때에는 producer 프로세스에서는 item을 생성하므로 생성된 아이템을 write해주고 (이 때 fprint가 쓰인다. ) consumer 프로세스서는 item을 사용하므로 (읽으므로) producer 프로세스가 생성한 아이템을 read 해준다.

- 부모 프로세스는 자식 프로세스의 수행이 완료될 때까지 wait()를 해준다. Wait()를 하지 않았을 경우 자식 프로세스는 주소 공간과 할당된 자원이 없어지지만 PCB가 유지되는 좀비상태가 된다.

**3.2 pthread 로 쓰레드를 생성하여 운영하는 방식**

- 스레드는 광역변수로 통신을 하기 때문에, 스레드 producer 과 consumer 간의 통신을 위해서 각각 스레드의 인덱스값을 광역변수로 넣어주었다. 또한 아이템 생성(write) 및 사용(read)도 광연변수 버퍼의 값을 바꿔주는 방식으로 하면 되기 때문에 프로세스에 비해 통신이 매우 수월하다.

- 스레드를 생성한 후 실행시켜 주었다면, 반드시 pthread\_join()을 해주어야 한다. 이는 프로세스의 wait() 와 비슷한 맥락으로, 스레드 하나의 수행이 먼저 끝나면 프로세스가 그대로 끝나버리기 때문에 먼저 끝난 스레드가 다른 스레드의 수행이 끝날 때까지 대기해줘야 하기 때문이다.