|  |
| --- |
| **201611229 장정아 운영체제 Producer-Consumer 문제** |
| **1. pthread의 mutex를 이용한 Producer-Consumer 구현** |
| #include <stdio.h>  #include <pthread.h>  #include <unistd.h>  void\* consumer(void\* arg);  int buffer[100];  int in = -1; // Producer스레드의 buffer index  int out = -1; // Consumer스레드의 buffer index  int counter = 0; // buffer 안의 원소의 개수  pthread\_mutex\_t lock = PTHREAD\_MUTEX\_INITIALIZER; // mutex 변수  pthread\_cond\_t bufffull = PTHREAD\_COND\_INITIALIZER;  pthread\_cond\_t buffempty = PTHREAD\_COND\_INITIALIZER;  int main() // main함수-producer 스레드  {  pthread\_t tid;  int num; int result;  pthread\_create(&tid, NULL, consumer, (void\*)&num); // consumer 스레드 생성  int produceritem = 1; // 생성하는 item의 번호  for (int i = 0; i < 1000; i++){ // 총 data set(추가할 수 있는 item의 개수)은 1000개  pthread\_mutex\_lock(&lock);  //\*\*\*\*\*\*\* 임계구역 \*\*\*\*\*\*\*  if (counter == 100){  ptrhead\_cond\_wait(&bufffull, &lock);  // buffer가 가득 찼을 경우에는 더 이상 buffer에 아이템을 채우지 않고 consumer 스레드가 buffer을 비우기를 기다림 -> 휴면 상태 }  in = (in + 1) % 100;  buffer[in] = produceritem;  printf("produce an item%d\n", produceritem);  // buffer에 produceritem 삽입 (생성)  sleep(1); // 문맥 교환이 발생!  produceritem = produceritem + 1;    counter = counter + 1;  // 독립적인 관계인 producer 스레드와 consumer 스레드는 counter이란 변수를 공유하므로 간섭이 발생할 수 있음 -> 임계구역 설정으로 방지!  pthread\_cond\_signal(&buffempty);  // buffer이 empty상태라 휴면에 들어갔던 consumer 스레드를 깨움  //\*\*\*\*\*\*\*\*\*\* 임계구역 종료 \*\*\*\*\*\*\*\*\*\*  pthread\_mutex\_unlock(&lock);  }  pthread\_join(tid, (void\*\*)&result); // consumer 스레드를 생성했으므로 consumer 스레드가 종료될 때까지 wait ( 좀비 스레드가 생성되는 것을 방지하기 위한 과정 ) }  void \*consumer(void\* arg){  int consumeritem; // consumer가 사용할 item  for (int i = 0; i < 1000; i++){  pthread\_mutex\_lock(&lock);  //\*\*\*\*\*\*\* 임계구역 \*\*\*\*\*\*\*  if (counter == 0){  pthread\_cond\_wiat(&buffempty, &lock);  } // buffer가 비었을 경우에는 더이상 buffer의 item을 소비하지 않고 producer 스레드가 buffer에 item을 채우기를 기다림    out = (out + 1) % 100;  consumeritem = buffer[out];  printf("consume an item%d\n", consumeritem); //buffer의 item을 사용  sleep(1); // 문맥 교환이 발생 !  counter = counter - 1; // 독립적인 관계인 producer 스레드와 consumer 스레드는 counter이란 변수를 공유하므로 간섭이 발생할 수 있음 -> 임계구역 설정으로 방지!  pthread\_cond\_signal(&bufffull);  // buffer이 full 상태라 휴면 상태였던 producer 스레드를 깨움  //\*\*\*\*\*\*\*\*\*\* 임계 구역 종료 \*\*\*\*\*\*\*\*\*\*\*  pthread\_mutex\_unlock(&lock);  }  } |
| **실행 결과** |
| …(반복)…  … |
| **2. pthread의 semaphore를 이용한 Producer-Consumer 구현** |
| #include <stdio.h>  #include <pthread.h>  #include <semaphore.h>  #include <unistd.h>  void\* consumer(void\* arg);  int buffer[100];  int in = -1; // Produceritem index  int out = -1; // Consumeritem index  sem\_t mutex, consume, produce;  int main(){ // main함수 - producer스레드  sem\_init(&mutex, 0, 1);// 공유변수 상호배제를 위한 자원 1개짜리 semaphore mutex  sem\_init(&produce, 0, 100);// item을 생성해 저장할 semaphore (producer 스레드)  sem\_init(&consume, 0, 0);// item을 사용할 semaphore (consumer 스레드)  pthread\_t tid;  int num; int result;  pthread\_create(&tid, NULL, consumer, (void\*)&num); // consumer 스레드 생성  int produceritem = 1;  for (int i = 0; i < 1000; i++){ //data set 1000개  printf("produce an item%d\n", produceritem);  sleep(1); // 문맥 교환 발생!  sem\_wait(&produce);  // item을 생성하는 과정이므로 produce 세마포어에 자원 요청 - produce 세마포어는 item이 생성될 때 자원이 줄어듦 = produce 세마포어에 빈 셀이 없을 때 item이 full (100개)    sem\_wait(&mutex);  // 자원이 오로지 1개인 세마포어변수 mutex - 1개 있던 자원이 요청되어 0개가 되었으므로 다른 스레드가 mutex 세마포어에 자원 요청을 할 경우 대기큐에 진입하게 됨 -> 임계 구역 지정  //\*\*\*\*\*\*\*\*\*\*\*\* 임계 구역 \*\*\*\*\*\*\*\*\*\*\*\*\*  in = (in + 1) % 100;  buffer[in] = produceritem;  produceritem = produceritem + 1; // buffer에 produceritem 저장 (생성)  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  sem\_post(&mutex); // 세마포어 mutex에 자원 반환 - 임계구역 end  sem\_post(&consume); // 새로운 item이 생성되었으므로 자원 생성(+1)을 알림  }  pthread\_join(tid, (void\*\*)&result);  // consumer 스레드가 종료되기 전에 producer 스레드가 종료되면 consumer 스레드는  좀비 스레드가 되어버림 : producer 스레드가 consumer 스레드의 종료까지 기다림 }  void \*consumer(void\* arg){  int consumeritem;  for (int i = 0; i < 1000; i++){ // data set 1000개  sem\_wait(&consume); // item을 소비하므로 consume 세마포어에 자원 요청  sem\_wait(&mutex);  //\*\*\*\*\*\*\*\*\*\*\*\* 임계 구역 \*\*\*\*\*\*\*\*\*\*\*\*\*  out = (out + 1) % 100;  consumeritem = buffer[out];  printf("consume an item%d\n", consumeritem);  // buffer에 존재하는 item 사용  sleep(1); // 문맥 교환 발생 !  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  sem\_post(&mutex); // 세마포어 mutex에 자원 반환 - 임계구역 end  sem\_post(&produce); // consumer 스레드가 item을 사용하여 빈 공간(produce에겐 자원)이 생겼으므로 자원 반환(+1)  }  } |
| **실행 결과** |
| …(반복)  … |
| **3. 정리** |
| **-** 독립적인 두 프로세스가 **공유 변수를 가짐에** **따라서 발생하는 간섭을** **해결**하기 위해서 등장  **: Mutex , Semaphore**  - Mutex와 Semaphore의 차이점   * Mutex : Busy-Waiting 알고리즘 : 프로세스A가 임계 구역에 진입했을 경우 뒤늦게 실행 상태에 도착한 프로세스B는 타임 슬라이스가 소진될 때까지 while( )문을 돈다. 타임 슬라이스가 소진되었다면 준비상태로 전환되고, 타임 슬라이스 안에 lock되었던 공유 자원을 얻었을 경우 실행된다. * Semaphore : Block/Wakeup 알고리즘 : 프로세스A가 임계 구역에 진입했을 경우 뒤늦게 도착한 프로세스B는 **대기큐로 이동해 대기 상태로 전환**된다.   - 생산자/소비자 문제에서의 Mutex   * Producer 스레드는 buffer이 full 상태라면 더 이상 item을 생성하지 않고, Consumer 스레드는 buffer이 empty 상태라면 더 이상 item을 사용하지 않는다. * Pthread\_cond\_wait( ) : 내부적으로 mutex를 잠금 해제하고 스레드 실행을 중지시킨다. * Pthread\_cond\_signal( ) : 조건 변수를 기다리고 있었던 스레드가 깨어난다. * 위의 함수를 통해 buffer의 상태에 따라 스레드를 대기시킬 수 있다. Buffer이 full이라면 Producer 스레드가 대기 상태로 진입, Consumer 스레드에서 item을 사용하기 전까지는 대기 상태를 유지한다. 반대로 Buffer이 Empty라면 Consumer 스레드가 대기 상태로 진입, Producer 스레드에서 item을 생성하기 전까지는 대기 상태를 유지한다. * 공유 변수인 counter을 변경 시킬때는 mutex로 임계구역을 설정해주어야 한다.   - 생산자/소비자 문제에서의 Semaphore   * 임계 구역을 설정하기 위한 Semaphore Mutex와 ( 자원 1개 )   Producer 스레드가 item을 저장하기 위한 공간 Semaphore Produce, ( 자원 100개 )  Consumer 스레드가 item을 사용하기 위한 공간 Semaphore Consume ( 자원 0개 )  가 사용되었다.   * Produce의 셀 하나는 item을 위한 “빈 공간”을 의미한다. 즉, Produce의 자원이 100개라면 item은 0개인 것과 같다. * Consume은 자원이 0개이므로 item이 생성되기 전까지 Consumer 스레드가 수행되었다면 대기큐에 들어가게 된다. * Item 생성시 Produce는 wait 연산을, Consume은 signal 연산을 하게 된다. * Item 사용시 Produce는 signal 연산을, Produce는 wait 연산을 하게 된다. |