운영체제 HW1.

201613989 김진산

1. Producer-Consumer Problem

1-1) 현재 수정전 코드에 대한 분석 내용을 기술하시오.

1-1-1) 코드 흐름

현재 코드는 생산자와 소비자를 나타내는 코드이다. 생산자는 아이템을 생성, 소비자는 아이템을 소비하는 역할을 한다. 사용자가 생산자 또는 소비자 프로세스 중 원하는 것을 선택하여 실행시킨다. 생산자를 실행하면 아이템 생산을, 소비자를 실행하면 아이템 소비를 진행한다.

1-1-2) 변수

-mutex : 생산자 또는 소비자가 실행되는 동안 다른 작업이 실행되지 않도록 상호 배제한다.

-x : 현재 가지고 있는 아이템의 개수

-full, empty: 두 변수의 값을 조정함으로써 아이템의 최대 생산 또는 소비를 제어한다. full이 0일경우 아이템이 존재하지 않으므로 소비자는 실행할수 없다. empty가 0일 경우 생산할 공간이 없으므로 생산자는 실행할수 없다.

1-1-3) 생산자 코드

1. 상호 배제를 시키기 위해 mutex값을 0으로 감소시킨다.

2. full값을 증가시켜 생산했음을 나타낸다.

3. empty값을 감소시켜 빈 공간이 1개 줄었음을 나타낸다.

4. 모든 작업이 끝났으므로 mutex값을 1로 증가시켜 다른 프로세스가 실행 가능하도록 한다.

1-1-4) 소비자 코드

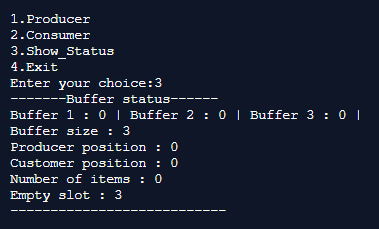
1. 상호 배제를 시키기 위해 mutex값을 0으로 감소시킨다.

2. full값을 감소시켜 소비했음을 나타낸다.

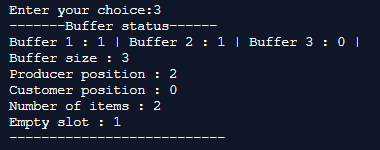
3. empty값을 증가시켜 빈 공간이 1개 생겼음을 나타낸다.

4. 모든 작업이 끝났으므로 mutex값을 1로 증가시켜 다른 프로세스가 실행 가능하도록 한다.

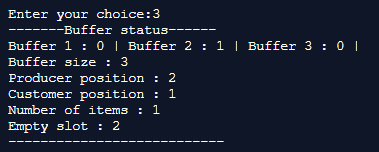
1-2) 이 코드를 이용하여, 교재 그림 5-12와 같이, 임계구역의 상태(총버퍼크기, producer, consumer 가 가리키는 위치, 아이템갯수, 빈 슬롯개수 등)가 출력되도록 하는 옵션을 코드에 추 가하고, 실행 결과 화면을 보이시오.



<메뉴 추가 및 임계구역 상태>



<생산자 2회 실행 후 임계구역 상태>



<생산자 2회, 소비자 1회 실행 후 임계구역 상태>

1-3) 표를 이용하여, 수정전 코드에 대비하여 수정한 내용을 기술하시오.

|  |  |
| --- | --- |
| 기능 추가를 위한 변수 변경 | |
| 수정 전 | 수정 후 |
| int mutex=1,full=0,empty=3,x=0; | #define buf\_size 3  int mutex=1,x=0;  //상호배재, 버퍼크기 제어 변수, 아이템 수  int p\_pos=0,c\_pos=0; //생산자, 소비자 위치  int\* buf;  //버퍼, 처음은 0으로 모두 초기화 |
| 변수 변경에 따른 생산자 진입조건 변경 | |
| 수정 전 | 수정 후 |
| if((mutex==1)&&(empty!=0)) | if((mutex==1)&&(x!=buf\_size)) |
| 변수 변경에 따른 소비자 진입조건 변경 | |
| 수정 전 | 수정 후 |
| if((mutex==1)&&(full!=0)) | if((mutex==1)&&(x!=0)) |
| 임계구역 표시 메뉴 생성 | |
| 수정 전 | 수정 후 |
| printf("\n1.Producer\n2.Consumer\n3.Exit"); | printf("\n1.Producer\n2.Consumer\n3.Show\_Status\n4.Exit"); |
| 임계구역 표시 메뉴 진입 생성 | |
| 수정 전 | 수정 후 |
| - | case 3: if(mutex==1)  show\_status();  break; |
| 임계구역 표시 함수 생성 | |
| 수정 전 | 수정 후 |
| - | void show\_status(){  int i;  mutex = wait(mutex);  printf("-------Buffer status------\n");  for(i=0; i<buf\_size; i++){  printf("Buffer %d : %d | ", i+1, buf[i]); // 버퍼의 상태 (1 = 아이템 있음, 0 = 아이템 없음)  }  printf("\n");  printf("Buffer size : %d\n",buf\_size); //총 버퍼 크기  printf("Producer position : %d\n", p\_pos); //생산자 위치  printf("Customer position : %d\n", c\_pos); //소비자 위치  printf("Number of items : %d\n", x); //아이템 수  printf("Empty slot : %d\n", buf\_size-x); //빈 슬롯 수  printf("---------------------------\n");  mutex=signal(mutex);  } |

|  |
| --- |
| 전체 코드 |
| #include<stdio.h>  #include<stdlib.h>  #define buf\_size 3  int mutex=1,x=0; //상호배재, 버퍼크기 제어 변수, 아이템 수  int p\_pos=0,c\_pos=0; //생산자, 소비자 위치  int\* buf; //버퍼, 처음은 0으로 모두 초기화  int main()  {  int n;  buf = (int\*)malloc(sizeof(int)\*buf\_size);  void producer();  void consumer();  void show\_status();  int wait(int);  int signal(int);  printf("\n1.Producer\n2.Consumer\n3.Show\_Status\n4.Exit");  while(1)  {  printf("\nEnter your choice:");  scanf("%d",&n);  switch(n)  {  case 1:  if((mutex==1)&&(x!=buf\_size))  producer();  else  printf("Buffer is full!!");  break;  case 2:  if((mutex==1)&&(x!=0))  consumer();  else  printf("Buffer is empty!!");  break;  case 3:  if(mutex==1)  show\_status();  break;  case 4:  exit(0);  break;  }  }  return 0;  }  int wait(int s)  {  return (--s);  }  int signal(int s)  {  return(++s);  }  void show\_status(){  int i;  mutex = wait(mutex);  printf("-------Buffer status------\n");  for(i=0; i<buf\_size; i++){  printf("Buffer %d : %d | ", i+1, buf[i]); // 버퍼의 상태 (1 = 아이템 있음, 0 = 아이템 없음)  }  printf("\n");  printf("Buffer size : %d\n",buf\_size); //총 버퍼 크기  printf("Producer position : %d\n", p\_pos); //생산자 위치  printf("Customer position : %d\n", c\_pos); //소비자 위치  printf("Number of items : %d\n", x); //아이템 수  printf("Empty slot : %d\n", buf\_size-x); //빈 슬롯 수  printf("---------------------------\n");  mutex=signal(mutex);  }  void producer()  {  mutex=wait(mutex); //상호 배제 on  if(p\_pos!=buf\_size) p\_pos++; //생산자의 위치 조정  else p\_pos = 1;  buf[p\_pos-1] = 1; // 버퍼에 값 추가  x++; //아이템 수 증가  printf("\nProducer produces the item %d\n",x);  mutex=signal(mutex); //상호 배제 off  }  void consumer()  {  mutex=wait(mutex); //상호 배제 on  buf[c\_pos] = 0; //버퍼에 값 제거  if(c\_pos!=buf\_size) c\_pos++; //소비자의 위치 조정  else c\_pos = 1;  printf("\nConsumer consumes item %d\n",x);  x--; //아이템 수 감소  mutex=signal(mutex); //상호 배제 off  } |

2. Dining Philosophers Problem

2-1) 현재 수정 전 코드에 대한 분석 내용을 기술하시오.

2-1-1) 코드 흐름

코드는 식사하는 철학자 문제를 나타내는 코드이다. 세마포어와 스레드를 이용하여 식사하는 철학자 문제를 표현한다.

철학자 5인의 각각의 philospher()를 실행시키는 스레드를 만든다. thinking이라는 메시지를 출력한다. 스레드를 만들 때 철학사의 상태는 eating으로 초기화 된다. 이는 스레드 생성과 동시에 test()함수를 통과하지 못하도록 방지하는 역할을 한다.

philospher()함수는 take\_fork와 put\_fork로 이루어져 있고 이를 반복한다. take\_fork와 put\_fork는 세마포어 mutex를 이용하여 실행 중 다른 프로세스의 접근을 막는다. test()함수가 끝나면 take\_fork는 세마포어 mutex를 증가시켜 다른 프로세스의 실행을 허가한다. 또 세마포어 S[]를 감소시키는데 이때 S[]가 이미 0이면 증가할 때까지 대기한다. 이는 test()코드의 if조건을 만족시키지 못하여 식사를 하지 못하여 포크를 잡지 못한 상태에서 put\_fork()함수로 진행을 막아 포크를 두는 행위를 막는다. 즉, S[]를 이용하여 손에 없는 포크를 내려두지 못하도록 막는 역할을 한다.

take\_fork는 포크를 집는 것으로 양 옆이 식사 중이 아니고, 해당 철학자가 배고픔 상태일 경우 식사를 하도록 한다. eating 메시를 출력 후 세마포어 S[]를 증가시켜 take\_fork()가 끝날 수 있도록 한다.

put\_fork는 포크를 두는 것으로 철학자의 상태를 thinking으로 변화시키고 양 옆의 철학자를 test()함수를 통해 식사가 가능한지 판단한다.

2-1-2) 변수

- N: 철학자의 수

- THINKING, HUNGRY, EATING: 철학자의 상태

- LEFT, RIGHT: 왼쪽, 오른쪽 표현 (철학자가 5명이므로 +4,+1후 %5)

- state[]: 철학자의 상태 표시

- phil[]: 철학자 표시

- mutex: 세마포어, 다른 작업이 동시에 실행되지 않도록 상호 배제

- S[]: 세마포어, 각각의 철학자들이 포크를 잡기 전에 두는 행동을 할 수 없도록 막는다.

2-1-3) 메인 함수

1. 스레드 배열 생성

2. mutex 세마포어 초기화 (초기값 1, 실행이 가능하다.)

3. S[]배열 세마포어 초기화 (초기값 0, 값이 post실행이 되야 실행가능, 그 전까지 대기)

4. 스레드 생성 (philoshoper()를 실행하는 스레드)

5. 스레드 종료 대기

2-1-4) philospher 함수

1. take\_fork (포크를 잡는 행위)

2. put\_fork (포크를 두는 행위)

3. 반복

2-1-5) take\_fork 함수

1. mutex 감소

2. 철학자의 상태 = eating 변경

3. test()함수 실행 (양 옆이 식사중인 상태가 아니면 상태 = 식사 중으로 변경)

4. mutex 증가

5. S[] 감소 (함수가 끝나서 put\_fork로 넘어가지 않도록 방지 - 즉 test코드를 실행하여 if문을 통과하여 eating으로 상태가 변하기 전까지는 put\_fork를 실행하지 않도록 방지한다.)

2-1-6) put\_fork 함수

1. mutex 감소

2. 철학자의 상태 = thinking 변경

3. test(왼쪽 철학자) 실행 (왼쪽의 양 옆이 식사인 상태가 아니면 왼쪽의 상태 = eating으로 변경)

4. test(오른쪽 철학자) (오른쪽의 양 옆이 식사인 상태가 아니면 왼쪽의 상태 = eating 변경)

5. mutex 증가

2-2) 이 코드를 이용하여 루프가 반복될 때 마다, 교재 그림 6-8처럼, 자원 할당 그래프의 형식으로 출력되도록 코드를 수정하고, 실행 결과 화면을 보이시오. (그림 6-5 자원 할당 그래프의 형식을 참고하시오. 즉, 프로세스와 자원간 할당, 기다림이 화살표로 표시되면 됨).

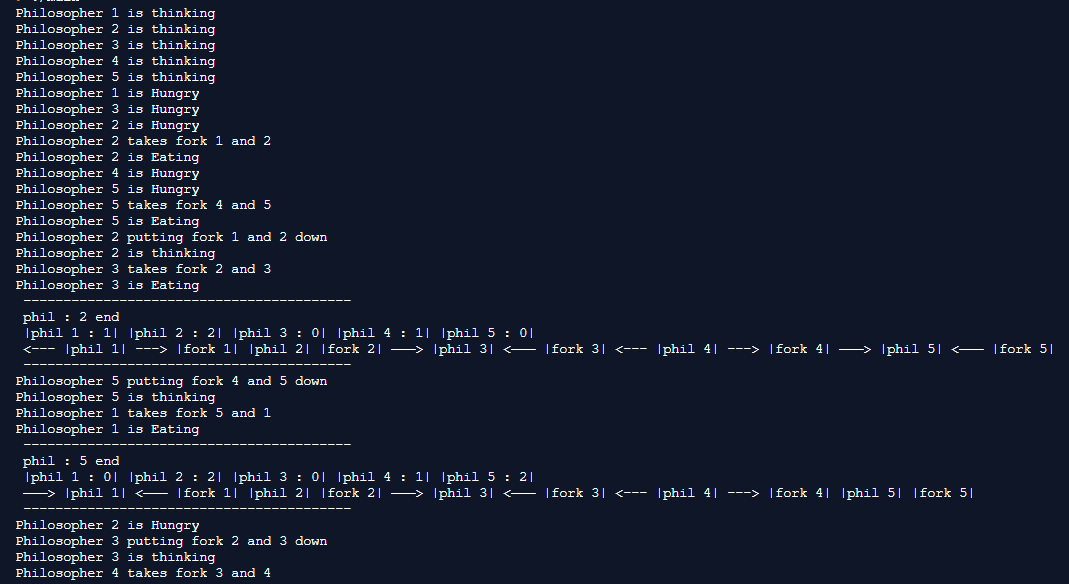
1. Philosopher 함수가 끝날 때 자원 할당 그래프의 형식으로 출력된다.

2. 구분을 위해 점선을 사용한다.

3. 어떤 philosopher 함수가 끝났는지 표기하고 철학자들의 상태를 코드 초기에 주어진 상태에 대한 숫자를 표기한다.

4. ──>는 자원 할당, --->는 자원 할당을 기다림을 표시한다.

5. hungry는 자원 할당 대기, eating은 자원 할당 상태, thinking은 자원 할당과 관련 없는 상태이다.



<Dining Philosophers Problem 실행 화면>

2-3) 표를 이용하여, 수정전 코드에 대비하여 수정한 내용을 기술하시오.

|  |  |
| --- | --- |
| 컴파일 오류로 인한 헤더파일 추가 | |
| 수정 전 | 수정 후 |
| - | #include <unistd.h> //sleep 사용을 위한 헤더파일 (repl.it에서 windows.h 오류) |
| 철학자의 상태를 보여주기 위한 함수 추가 | |
| 수정 전 | 수정 후 |
| - | void show\_state(){  for(int i = 0; i<N; i++){  printf(" |phil %d : %d|",i+1, state[i]);  }  printf("\n");  } |
| 자원할당 그래프를 나타내기 위한 함수 추가 | |
| 수정 전 | 수정 후 |
| - | void show\_graph(int i){  sem\_wait(&mutex);  printf(" -----------------------------------------\n");  printf(" phil : %d end\n", i+1);  show\_state();  for(int i = 0; i<N; i++){  int r = (i+1)%5;  if(state[i] == 0){  printf(" ───> |phil %d| <─── ",i+1);  }else if(state[i] == 1){  printf(" <--- |phil %d| ---> ",i+1);  }else if(state[i] == 2){  printf(" |phil %d| ",i+1);  }  printf("|fork %d|", i+1);  }  printf("\n");  printf(" -----------------------------------------\n");  sem\_post(&mutex);  } |
| Philosopher 함수 끝에 show\_graph()함수 추가 | |
| 수정 전 | 수정 후 |
| void\* philospher(void\* num)  {  while (1) {  int\* i = num;  sleep(1);  take\_fork(\*i); //포크 잡기  sleep(0);  put\_fork(\*i); //포크 두기  }  } | void\* philospher(void\* num)  {  while (1) {  int\* i = num;  sleep(1);  take\_fork(\*i); //포크 잡기  sleep(0);  put\_fork(\*i); //포크 두기  show\_graph(\*i);  }  } |

|  |
| --- |
| 전체 코드 |
| #include <pthread.h> // 스레드 헤더파일  #include <semaphore.h> //세마포어 헤더파일  #include <stdio.h>  #include <unistd.h>//sleep 사용을 위한 헤더파일 (repl.it에서 windows.h 오류)  #define N 5 //철학자 수  #define THINKING 2 //철학자 상태  #define HUNGRY 1 //철학자 상태  #define EATING 0 //철학자 상태  #define LEFT (phnum + 4) % N // 철학자 수가 5명이므로 왼쪽  #define RIGHT (phnum + 1) % N // 철학자 수가 5명이므로 오른쪽  int state[N]; //철학자의 상태  int phil[N] = { 0, 1, 2, 3, 4 };  sem\_t mutex; //상호 배제  sem\_t S[N]; //세마포어  void show\_state(){  for(int i = 0; i<N; i++){  printf(" |phil %d : %d|",i+1, state[i]);  }  printf("\n");  }  void show\_graph(int i){  sem\_wait(&mutex);  printf(" -----------------------------------------\n");  printf(" phil : %d end\n", i+1);  show\_state();  for(int i = 0; i<N; i++){  int r = (i+1)%5;  if(state[i] == 0){  printf(" ───> |phil %d| <─── ",i+1);  }else if(state[i] == 1){  printf(" <--- |phil %d| ---> ",i+1);  }else if(state[i] == 2){  printf(" |phil %d| ",i+1);  }  printf("|fork %d|", i+1);  }  printf("\n");  printf(" -----------------------------------------\n");  sem\_post(&mutex);  }  void test(int phnum)  {  if (state[phnum] == HUNGRY && state[LEFT] != EATING  && state[RIGHT] != EATING) {  // state that eating  state[phnum] = EATING;  sleep(2);  printf("Philosopher %d takes fork %d and %d\n",  phnum + 1, LEFT + 1, phnum + 1);  printf("Philosopher %d is Eating\n", phnum + 1);  // sem\_post(&S[phnum]) has no effect  // during takefork  // used to wake up hungry philosophers  // during putfork  sem\_post(&S[phnum]); //세마포어 증가  }  }  // take up chopsticks  void take\_fork(int phnum)  {  sem\_wait(&mutex); //세마포어 감소  // state that hungry  state[phnum] = HUNGRY; //배고픔 상태로 변경  printf("Philosopher %d is Hungry\n", phnum + 1);  // eat if neighbours are not eating  test(phnum); //양옆에서 먹고 있는 경우 세마포어 증가  sem\_post(&mutex); //세마포어 증가  // if unable to eat wait to be signalled  sem\_wait(&S[phnum]); //세마포어 감소  sleep(1);  }  // put down chopsticks  void put\_fork(int phnum)  {  sem\_wait(&mutex); //세마포어 감소  // state that thinking  state[phnum] = THINKING; //생각상태로 변경  printf("Philosopher %d putting fork %d and %d down\n",  phnum + 1, LEFT + 1, phnum + 1);  printf("Philosopher %d is thinking\n", phnum + 1);  test(LEFT);  test(RIGHT);  sem\_post(&mutex); //세마포어 증가  }  void\* philospher(void\* num)  {  while (1) {  int\* i = num;  sleep(1);  take\_fork(\*i); //포크 잡기  sleep(0);  put\_fork(\*i); //포크 두기  show\_graph(\*i);  }  }  int main()  {  int i;  pthread\_t thread\_id[N]; //스레드형 배열  sem\_init(&mutex, 0, 1); // initialize the semaphores  for (i = 0; i < N; i++) // initialize the semaphores  sem\_init(&S[i], 0, 0);  for (i = 0; i < N; i++) { // create philosopher processes  pthread\_create(&thread\_id[i], NULL,philospher, &phil[i]);  printf("Philosopher %d is thinking\n", i + 1);  }  for (i = 0; i < N; i++) // 쓰레드 종료 대기  pthread\_join(thread\_id[i], NULL);  } |