chapter14. Critical Sections and Semaphores

objectives

1. Learn about semaphores and their properties
2. Experiment with synchronization
3. Explore critical section behavior
4. Use POSIX named and unnamed semaphores
5. Understand semaphore management

Critical sections

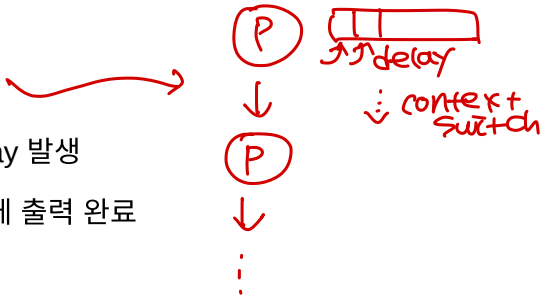
반드시 mutually exclusive(상호배제)로 실행되어야 하는 code segment

→ shared device : 한 번에 한 process만 접근해야 하는 exclusive resources

- **Program 14.1** → wrong example
  - generate process chain + print 한 번에 한 문자
  - 각 문자가 출력된 후에 command line마다 추가적인 delay 발생
    - 보통은 delay가 짧아서 context-switch 일어나기 전에 출력 완료
    - 꽤 많은 양의 delay

      → 각 process가 오직 한 개의 문자만 출력

      → 여러 process output 섞임




```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>
#include "restart.h"
#define BUFSIZE 1024

int main(int argc, char *argv[]) {
   char buffer[BUFSIZE];
   char *c;
```

```
    pid_t childpid = 0;
    int delay;
    volatile int dummy = 0;
    int i, n;

    if (argc != 3){   /* check for valid number of command-line arguments */
       fprintf (stderr, "Usage: %s processes delay\n", argv[0]);
       return 1;
    }
    n = atoi(argv[1]);
    delay = atoi(argv[2]);
    for (i = 1; i < n; i++) → 자식 process 생성
       if (childpid = fork())
          break;
    snprintf(buffer, BUFSIZE,  → Ø 돈가
        "i:%d  process ID:%ld  parent ID:%ld  child ID:%ld\n",
        i, (long)getpid(), (long)getppid(), (long)childpid);

     c = buffer; ⇒ 시작 pointer → 하나씩 출력하면서 움직임.
    /********************** start of critical section **********************/
    while (*c != '\0') {
       fputc(*c, stderr);
       c++;
       for (i = 0; i < delay; i++)
          dummy++;
    }
    /********************** end of critical section ************************/
    if (r_wait(NULL) == -1)
       return 1;
    return 0;
}
```

critical 이어야 하는데 X → 여러 process가 동시에 수행

- **section**
  1. Entry section : shared variable이나 다른 resource에 대해 허가를 요청하는 부분
     a. 허가 받지 못 한 process → waiting queue에서 대기
  2. Critical section : 한 process가 이미 진입했을 때 다른 process는 진입 x
  3. Exit section : lock을 얻은(허가를 받은) process를 release
     a. next thread가 lock을 받을 수 있는지 entry section에서 확인 후 lock 넘겨줌
  4. Remainder section : access에 대한 releasing 이후 남은 code execute

- **solution to critical-section problem**
  1. Mutual exclusion
     a. 만약 process P가 critical section 내에 있다면 → 다른 process 접근 불가
  2. progress
     a. 만약 critical section 내에 있는 process가 없고

critical section에 들어가길 원하는 process가 있다면

(waiting queue가 비어있지 않다면)

→ reminder section에서 실행하지 않고 있는 process들만 critical section에 들어갈 다음process를 결정하는데 참여 가능

→ 한 process가 critical section을 빠져 나오면 해당 process는 unlock하고 다음 process가 waiting queue에서 빠져나와서 언젠가는 꼭 진입 가능해야 함.

3. bounded waiting
   a. waiting process의 경우, 기다리는 시간이 한정적이어야 함

      → 제한된 시간 내에는 꼭 진입할 수 있어야 함

Semaphores

two atomic operation을 수행할 수 있는 정수형 변수

→ OS kernel에서 관리(high-level management of mutual exclusion and sychronization)

↳ atomic하게 수행되도록 보장

- operation (critical section)
  1. wait : semaphore 감소
     a. S>0 → S 감소
     b. S=0 → 호출한 함수 block

     ↳ 기본적으로 semaphore는 양수

```
void wait(semaphore_t *sp){
  if( sp->value > 0 )
    sp->value--;
  else{
    <add this thread to sp->list>
    <block>
  }
}
```

  2. signal : semaphore 증가
     a. S에 대해 thread가 block → waiting thread 중 하나를 unblock
     b. block된 thread = 0 → S 증가

```
void signal(semaphore_t *sp){
  if( sp->list != NULL )
    <remove a thread from sp->list>
  else
```

```
    sp->value++;
}
```

- **example**

1. Critical section using semaphore : 어떤 값으로 초기화하냐가 중요 ⇒ semaphore의 초기값 : shared resource 할 수 있는 available process 개수

```
wait(&S);
<critical section>
signal(&S);
<remainder section>
```

→ S=0으로 초기화

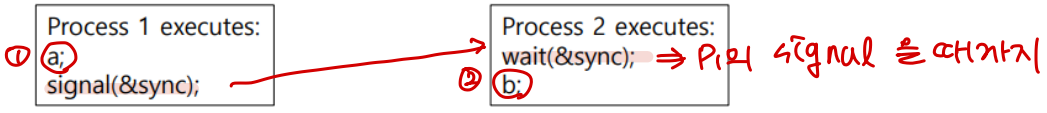
: 모든 wait(&S) block 후 signal 을 호출하는 다른 process가 없다면 deadlock

→ S=8으로 초기화

: ciritcal sectin에 8이 0이 될 때까지 진입 가능(총 8개의 process)

→ 더 이상 critical이 아님. 8개의 process만 진입할 수 있는 새로운 동기화

2. enforcing the order of operations (수행 순서의 제한)



① sync = 0 초기화 → process2가 process1이 signal을 호출할 때까지 wait에서 block

3. semaphore 2개 사용

① S=1, Q=1로 초기화

: 둘 중 하나가 먼저 wait 실행

→ iteration 횟수가 하나 이상 벌어지지 않도록 실행됨

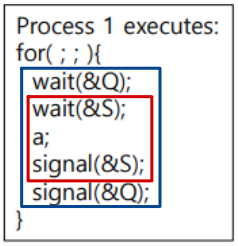
② 하나는 0, 하나는 1로 초기화

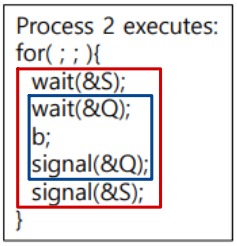
: process가 번갈아 실행됨

③ S=0, Q=0으로 초기화

: deadlock (wait가 먼저 호출되어 두 process block)

4. semaphore 2개 사용 ver.2





① S=1, Q=1로 초기화

: CPU가 선택한 process가 무엇이냐에 따라 상황이 다름

→ 동기화될 수도 있고 deadlock에 빠질 수도 있음

↳ wait wait 사이에 P2가 실행될 경우 P1, P2 모두 deadlock

POSIX:SEM Unnamed Semaphores

```
#include <semaphore.h>
sem_t sem;

//sem_t : semaphore type -> 정수값
//       supports POSIX:SEM semaphores if it defines _POSIX_SEMAPHORES in unistd.h
```

- **Initialization / Destroy**

```
#include <semaphore.h>

int sem_init(sem_t *sem, int pshared, unsigned value);
//사용하기 전에 초기화 해야 함
//sem을 value로 초기화

//sem : 초기화 될 semaphore
//pshared : 공유 여부
//    -> 0 : 초기화한 process의 threads만 공유
//    -> nonzero : 'sem'에 접근할 수 있는 모든 process가 공유
//value : 초기화 값 -> 음수 불가능

int sem_destroy(sem_t *sem);
//destroy 함수
          ↳ 다쓰고 release

//return 0 -> successful
//return -1 -> unsuccessful
```

- **Operations**

```
#include <semaphore.h>

int sem_post(sem_t *sem);
//signal 함수 -> signal safe + signal handler에 의해 호출됨

int sem_wait(sem_t *sem);
//wait 함수

int sem_trywait(sem_t *sem);
//sem_wait의 비동기식 함수
//0인 semaphore를 decrement하려고 할 때 block X
// -> errno를 EAGAIN으로 설정 + return -1

int sem_getvalue(sem_t *restrict sem, int *restrict sval);
//semaphore의 값을 검사하도록 허가해주는 함수

//sval : semaphore에게 영향을 주지 않고 기존 semaphore 값을 가지고 있음
//     -> but, 어느 시점에 return 하는지에 따라서 값이 변경될 수 있음

//return 0 -> successful
//return -1 -> unsuccessful
```

- **example**

```
#include <errno.h>
#include <semaphore.h>
static int shared = 0;
static sem_t sharedsem;

int initshared(int val) {
  if (sem_init(&sharedsem, 0, 1) == -1)
    return -1;
  shared = val;
  return 0;
}

//return the current value
int getshared(int *sval) {
  while (sem_wait(&sharedsem) == -1)
    if (errno != EINTR)
      return -1;
    *sval = shared;
    return sem_post(&sharedsem);
}

//increment
int incshared() {
  while (sem_wait(&sharedsem) == -1)
    if (errno != EINTR)
      return -1;
  shared++;
  return sem_post(&sharedsem);
}
```

POSIX:SEM Named Semaphores

memory를 공유하지 않고 process를 synchronize

- name, userID, groupID, permissions → file처럼
- name : slash로 시작

  → 같은 name의 semaphore를 여러 process, threads에서 open할 경우 같은 semaphore

  → POSIX : slash로 시작하지 않는 같은 이름의 semaphore를 참조하도록 define X

*named semaphore는 다른 process에서 보이지는 X

- **create and opening**

```
#include <semaphore.h>
sem_t *sem_open(const char *name, int oflag, …);
//named semaphore와 sem_t value를 연결하는 함수

//name : named semaphore → open 하고 싶은 semaphore 이름
//oflag : create or access 결정
//  -> O_CREAT : 이름이 같은 기존 semaphore가 있다면 기존 것 open
//             mode, value parameter가 추가로 필요 ← 새로 생성 시 !
//  -> O_EXCL + O_CREAT : 이름이 같은 semaphore가 이미 있다면 return error

//return 0 -> successful
//return -1 -> unsuccessful
```

+) mode : semaphore는 실행 권한 X (읽기, 쓰기만 가능)

permission → 초기값 설정

- **example1 - named semaphore open, create해서 access**

```
#include <errno.h>
#include <fcntl.h>
#include <semaphore.h>
#include <sys/stat.h>
#define PERMS (mode_t)(S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)
#define FLAGS (O_CREAT | O_EXCL)

int getnamed(char *name, sem_t **sem, int val) {
   while (((*sem = sem_open(name, FLAGS , PERMS, val)) == SEM_FAILED) &&
           (errno == EINTR)) ;
   if (*sem != SEM_FAILED)
       return 0;
   if (errno != EEXIST)
      return -1;
   while (((*sem = sem_open(name, 0)) == SEM_FAILED) && (errno == EINTR)) ;
   if (*sem != SEM_FAILED)
       return 0;
   return -1;
}
```

① create

새로 만들 때 필요

성공하면 0을 return

다시 보통

② open

기존의 semaphore open

- **example2 - named semaphore로 program 14.1 해결**

```
#include <errno.h>
#include <semaphore.h>
```

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include "restart.h"
#define BUFSIZE 1024
int getnamed(char *name, sem_t **sem, int val);

int main  (int argc, char *argv[]) {
   ...
   while (sem_wait(semlockp) == -1)                          /* entry section */
       if (errno != EINTR) {
          perror("Failed to lock semlock");
          return 1;
       }
   while (*c != '\0') {                                      /* critical section */
      fputc(*c, stderr);
      c++;
      for (i = 0; i < delay; i++)
         dummy++;
   }
   if (sem_post(semlockp) == -1) {                           /* exit section */
      perror("Failed to unlock semlock");
      return 1;
   }
   if (r_wait(NULL) == -1)                                   /* remainder section */
      return 1;
   return 0;
}
```