CS330 HW1

team 27 :
20150109 김동균
20150195 김정우

Question 1

가정:
버퍼와 세마포어의 연산을 수행해주는 시스템이 존재한다.
cook은 초기값 0인 세마포어 sema_cook을 가지고 있고 각 customer는 초기값 0인 세마포어 sema_customer를 가지고 있다.
크기 n의 전역 버퍼 store가 있어 가게 안에 앉아 있는 customer들을 나타낸다.
크기 무제한의 전역 버퍼 orders_list가 있어 customer들의 주문 내역을 저장한다.
cook은 쉬지 않고 일한다.

```
cook:
      while (true):
            // 주문 내역을 확인하여, 케밥을 줄 customer가 없으면 쉰다.
            sema_down (sema_cook)
            // customer가 남았다면 customer에게 줄 케밥을 만든다.
            making_kebab ()
            // 주문 내역 하나를 빼고, customer에게 케밥이 완성되었음을 알린다.
            sema_up (pop_front (orders_list))

customer:
      // 가게가 꽉 차있으면 포기한다.
      if store is full:
            leave_without_kebab()

      // 가게에 들어가며, 주문 내역을 작성한 후 cook에게 준다.
      insert (itself at store)
      push_back (itself at orders_list)
      sema_up (sema_cook)

      // 케밥이 나올 때까지 대기한다.
      sema_down (sema_customer of itself)

      // 케밥을 가지고 나간다.
      getting_kebab ()
      remove (itself from store)
```

Question 2

```
struct lock {
      int is_blocked;   // 대기 중인 프로세스가 있으면 1, 없으면 0. 0으로 초기화.
};

void acquire(struct lock *lock) {
      while (!compare_and_swap(&lock->is_locked, 0, 1));
}

void release(struct lock *lock) {
      compare_and_swap(&lock->is_locked, 1, 0);
}
```

Question 3

0. System Layout

시스템 위에 여러 프로세스가 동작한다. 시스템의 컨트롤과 각 프로세스의 컨트롤은 시스템 모드와 프로세스 모드로 분리되어 있으며 모드 전환 시 서로의 레지스터 셋과 메모리 공간의 컨택스트 스위칭이 일어난다.
모든 시단위는 tick이며, 매 tick 시작 시 시스템이 특정 이벤트의 종료 신호나 타임아웃 등을 참고하여 프로세스를 운영하며 이 운영과정이 끝난 후 프로세스 모드로 전환되어 프로세스의 작업을 수행한다.
모드 전환이 매 tick마다 일어나므로, 한 tick의 길이는 시스템의 tick별 운영시간이 tick의 매우 일부를 차지하도록 정의해야 한다.

프로세스는 시스템의 운영을 담당하는 시스템 프로세스와 유저의 작업을 수행하는 유저 프로세스로 나뉜다. 시스템 프로세스들은 다음과 같으며, 이들은 부팅 후부터 시스템이 종료할 때까지 항상 존재한다.

- system_console - 유저와의 소통을 담당한다.
- system_creator - 프로세스의 생성을 담당한다.
- system_deletor - 프로세스의 제거를 담당한다.
- system_event_handler - 특정 이벤트의 처리를 담당한다.

프로세스가 부를 수 있는 이벤트의 종류와 개수가 정해져 있으며, system_event_handler는 각 이벤트마다 독립적으로 존재한다. 이벤트들은 필요에 따라 추가하면 된다. 예를 들어 wait_for_process라는 이벤트는 한 프로세스의 종료를 기다리는 이벤트라고 하자. 이때 system_event_handler[wait_for_process]는 기다릴 프로세스의 식별자를 받아 그 프로세스가 종료됨을 확인하고 시스템에게 이벤트 종료 신호를 보낸다.

메모리 공간은 크게 시스템 공간과 프로세스 공간으로 나뉘며, 프로세스는 시스템 공간은 물론 다른 프로세스 공간에 접근할 수 없다. 모든 자료의 공유는 시스템에 의해 수행된다.

1. Data Structures

모든 프로세스는 동일한 자료구조로 이루어져 있다.

- pid : 프로세스 식별자. 시스템 프로세스는 음의 정수를 가지고, 유저 프로세스는 양의 정수를 가진다.
- status: 현재 스케줄링 상태. 다음 중 하나이다.
  - CREATING: 생성 중
  - READY : CPU 사용 준비됨
  - RUNNING : CPU 사용 중
  - WAITING : 특정 이벤트 대기 중

DELETING : 제거 중

- priority : 스케줄링 우선순위. 1부터 10까지 존재한다.
- reg_set : 레지스터 집합. PC, SP, AC 등 모든 레지스터를 포함한다.
- pagedir : 페이지 디렉토리. 할당된 페이지 디렉토리 주소를 나타낸다.
- acq_io : 사용하고 있는 io 목록. I/O 장치, 파일 등

프로세스 스케줄링에 필요한 자료구조는 하나의 포인터와 4종류의 큐이다.

- running_process : 현재 CPU를 사용 중인 프로세스를 가리키는 포인터이다.
- ready_queue : READY 상태인 프로세스들의 큐이다.
- event_queue : 특정 이벤트를 대기중인 프로세스들의 큐이며, 이벤트마다 독립적으로 존재한다.
- creating_queue : 프로세스 생성을 위한 실행파일 정보들의 큐이다.
- deleting_queue : 삭제 중인 프로세스들의 큐이다.

2. Scheduling Policy

ready_queue 안의 모든 프로세스는 돌아가면서 CPU를 사용하며, 한 번에 자신의 고유한 priority만큼의 시간을 사용할 수 있다. 예를 들어 타임 슬라이스를 1 tick이라 하고 priority가 5라면 자신의 차례가 되었을 때 5tick 만큼 사용가능하며, 그 후 시스템에 의해 preempt되어 다음 프로세스에게 CPU를 양보한다.

3. System Calls

모든 프로세스는 시스템 콜을 통해 시스템이나 시스템 프로세스에게 여러 작업을 맡길 수 있다. 시스템 콜을 호출하면 프로세스는 시스템에게 해당하는 신호를 보내며, 시스템은 다음 tick에서 이 신호를 받아 자신이 작업을 처리하거나 해당하는 시스템 프로세스에 작업을 위임한다.

- halt : 시스템을 종료하기 위한 시스템 콜. 호출하면 시스템은 종료 루틴을 수행한다.

- process_create : 프로세스를 생성하기 위한 시스템 콜. 호출하면 시스템은 실행파일에 대한 정보를 creating_queue의 back에 push한 후 system_creator에게 생성 작업을 위임한다.

- process_yield : 다음 프로세스로 전환하기 위한 시스템 콜. 호출하면 시스템은 process_yield를 호출한 함수를 READY 상태로 설정한 후 ready_queue의 back에 push한다. 그 후 ready_queue의 front에 위치한 프로세스를 pop하여 RUNNING

상태로 설정하고 running_process를 그 프로세스로 바꾼다. 컨택스트 스위칭이 끝나면 프로세스 모드로 모드 전환하여 다음 프로세스에게 컨트롤을 넘긴다.

- process_call_event : 특정 이벤트를 호출하기 위한 시스템 콜. 호출하면 시스템은 우선 이벤트를 호출한 프로세스의 상태를 WAITING으로 바꾼다. 그 후 해당하는 event_queue의 back에 push하여 system_event_handler에게 이벤트 처리 작업을 위임한다.

- process_delete : 프로세스를 제거하기 위한 시스템 콜. 호출하면 시스템은 우선 process_delete를 호출한 함수를 DELETING 상태로 바꾼다. 그 후 deleting_queue의 back에 push하여 system_deletor에게 제거 작업을 위임한다. 물론 시스템 프로세스의 제거는 불가하다.

4. Process Cycle

새 프로세스의 실행은 system_console이 콘솔을 통해 실행파일에 대한 정보를 받은 후 process_create를 호출하여 처리된다.

새로 실행된 프로세스의 생성은 system_creator에 의해 creating_queue의 front부터 진행된다. system_creator가 프로세스 생성을 완료하면 해당 프로세스의 상태를 READY로 바꾼 후 ready_queue의 back에 push한다. 만약 creating_queue가 비어 있다면 바로 process_yield를 호출하여 다음 프로세스로 넘긴다.

프로세스의 제거는 system_deletor에 의해 deleting_queue의 front부터 진행된다. system_deletor가 프로세스 삭제를 완료하면 관련된 모든 정보가 시스템에서 삭제된다. 제거될 프로세스가 없다면 바로 process_yield를 호출하여 다음 프로세스로 넘긴다.

이벤트 처리는 각각의 system_event_handler가 처리하며, event가 끝나면 해당 event_queue의 모든 프로세스를 pop하여 READY 상태로 바꾼 후 ready_queue의 back에 순서대로 push한다. 호출된 event가 없으면 바로 process_yield를 호출하여 다음 프로세스로 넘긴다.

5. Thread System

스레드는 한 프로세스 당 여러 개를 가질 수 있으며, thread_create 시스템 콜에 의해 생성된다. 각 스레드는 프로세스에 할당된 tick을 n분하며, 만약 1 tick이 할당된 프로세스에 세 스레드가 존재하여 한 스레드가 1 tick을 사용하였다면 다음 차례와 다다음 차례의 1 tick은 나머지 두 스레드가 각각 사용한다.

스레드는 같은 프로세스 내의 모든 스레드와 같은 메모리 공간을 공유한다.

한 프로세스 내의 각 스레드 상태와 레지스터 집합, 스택은 독립적이다.

스레드 시스템을 구성하기 위해 프로세스 구조에 스레드들의 리스트 threads를 추가하고, 독립적인 요소인 status와 reg_set을 제거하자. 스레드의 자료구조는 다음과 같이 만든다.

- tid : 스레드 식별자.
- status : 스레드 상태.
- reg_set : 레지스터 집합.
- process : 자신이 속한 프로세스. 나머지 모든 자원은 프로세스 안에서 공유하며, 이 포인터를 통해 프로세스의 자원을 사용할 수 있다.