2022-Spring
운영체제

학번	2015150220
이름	이윤수

[OS Practice 2. Inter Process Communication]

Assignment 1-2. semaphore

Question :

특정 작업이 원자적으로 수행되도록 s_quit(), s_wait()를 사용, 사용되어야 하는 위치 와 이유?

Explanation

```
126 int produce(MessageBuffer **buffer, int sender_id, int data, int account_id) {
127 
128     /*----------------------------------------*/
129     /* TODO 3 : produce message               */
130     s_wait();
131     (*buffer)->messages[account_id].sender_id = sender_id;
132     (*buffer)->messages[account_id].data += data;
133 
134     if((*buffer)->messages[account_id].data<0){
135         printf("Balance is not sufficient\n");
136         (*buffer)->messages[account_id].data -= data;
137         s_quit();
138         return 0;
139     }
140 
141     (*buffer)->account_id = account_id;
142     (*buffer)->is_empty = 0;
143 
144     /* TODO 3 : END                           */
145     /*----------------------------------------*/
146     printf("produce message\n");
147     return 0;
148 }
149 
150 int consume(MessageBuffer **buffer, Message **message) {
151     if((*buffer)->is_empty)
152         return -1;
153 
154     /*----------------------------------------*/
155     /* TODO 4 : consume message               */
156     *message = &(*buffer)->messages[(*buffer)->account_id];
157     (*buffer)->is_empty=1;
158     s_quit();
159     /* TODO 4 : END                           */
160     /*----------------------------------------*/
161 
162     return 0;
163 }
```

먼저, produce() 함수의 첫번째 부분(130 번째 줄)에 s_wait()를 위치시켰습니다. 그리고 s_quit() 함수는 계좌 잔고 부족으로 produce 메시지 생성이 거절되는 부분(137 번째 줄)과, consume() 함수의 return 직전 마지막 부분(157 번째 줄)에 위치시켰습니다. 그 이유는 다음과 같습니다.

(1) 공유자원에 대한 접근

: 이 코드에서 공유자원은 init_buffer()를 통해서 만들어진 shared memory 입니다. 이때 이 공유자원에 직접 접근하여 shared momory 의 값에 접근하는 함수는 produce 함수와 consume() 함수라고 판단했습니다. Produce() 함수는 버퍼에 직접 값을 삽입, 수정하고 있고, consume() 함수는 버퍼의 is_empty 값이 0 이 되면, 포인터에 삽입 또는 수정된 message 를 할당합니다. 따라서 프로세스들 간에 CPU 자원을 두고 경쟁적인 환경(Race Condition)이 형성될 경우, 한 프로세스가 다른 프로세스에 의해 produce 된 메시지를 출력하는 오류가 발생할 수 있습니다. 그래서 처음 메시지를 생산하는 시작부분부터 생산된 메시지를 출력하는 부분까지를 위험지역(critical section)으로 설정하고 이에 대한 상호 배제(mutual exclusion)을 구현해야 한다고 생각했습니다.

(2) 구체적인 동작

: 최초로 설정된 세마포어의 값은 1 입니다(sem_union.val). 이때 producer.c 를 실행하는 즉, 메시지를 생산하는 서로 다른 두 개의 프로세스 A,B 가 있다고 가정합니다. 이때 A,B 는 attach buffer 과정에서 같은 semid 를 공유하는 것을 전제로 합니다. 먼저 수행되는 프로세스 A 는 s_wait()를 통과하는 과정에서 세마포어 값을 1 감소시켜 그 값이 0 이 됩니다. 이로 인해 나중에 도착하는 프로세스 B 는 s_wait()를 통과하지 못하고 대기합니다. 프로세스 A 에 의해 만들어진 메시지가 최종적으로 consume() 함수에서 메시지 포인터에 할당된 다음 s_quit()을 해주어야, 세마포어 값이 다시 1 로 증가되고, 기다리던 프로세스 B 가 위험지역에 진입합니다. (또는 계좌 잔액 부족으로 produce 함수 내에서 메시지 생성이 거절된 다음 s_quit()을 해줍니다.) 이를 통해 먼저 실행된 프로세스 A 가 produce 함수를 완전히 수행하고 그에 따라 의도된 메시지(잔액부족 혹은 거래 내역)의 출력이 완료된 이후에 다음 프로세스인 B 가 수행됩니다.

참고자료:
‘세마포어’ retrieved from https://mobilenuri.tistory.com/99.