

# 운영체제론 중간고사'20

소프트웨어학부

2020년 5월 7일

1. [40 pts] 다음 글을 읽고 빈 칸에 알맞는 말을 채워라.

- (a) 실시간 시스템에서 사건이 발생했을 때, 이를 컴퓨터가 처리하는 과정에서 인터럽트 지연과 디스패치 지연이 발생한다. 인터럽트 지연은 \_\_\_\_\_ 때문에 생기고, 디스패치 지연은 \_\_\_\_\_ 때문에 생긴다.
- (b) 코드, 데이터, 스레드 제어 블록, 열린 파일, 파일시스템, 런타임 스택, 시그널 중에서 스레드가 공유할 수 없는 것을 모두 고르면 \_\_\_\_\_이다.
- (c) 우선순위가 높은 프로세스가 낮은 프로세스가 소유하고 있는 리소스를 기다리면서 실행 순서가 뒤로 밀리는 현상을 \_\_\_\_\_(이)라고 한다.
- (d) 다음에 있는 OpenMP 코드는 \_\_\_\_\_개의 스레드를 생성한다.

```
#pragma omp parallel for
for (i = 0; i < N; i++) {
    c[i] = a[i] * b[i];
}
```

- (e) 다중프로세서 환경에서 로드 밸런싱은 캐시 미스를 줄이기 위해 \_\_\_\_\_을(를) 최대한 유지하는 방향으로 이루어져야 한다.
- (f) 지연된 스레드 취소 방식에서 스레드 취소 여부를 확인할 때 사용하는 POSIX 시스템 함수는 \_\_\_\_\_이다. 프로그램의 흐름이 이 함수를 호출한 지점으로 되돌아오지 않았다면 이 스레드가 \_\_\_\_\_되었다는 것을 의미한다.
- (g) \_\_\_\_\_시스템 호출은 새로운 프로세스를 생성하지 않고 현재 실행 중인 프로세스 공간을 새로운 프로그램으로 덮어씌우기 위해 사용한다.
- (h) SJF (Shortest Job First) 알고리즘의 preemptive 버전을 \_\_\_\_\_(이)라고 부른다.
- (i) 리눅스, 맥OS, 윈도우 같은 대부분의 상용 운영체제는 사용자 스레드와 커널 스레드를 \_\_\_\_\_매핑하는 방식을 사용한다.
- (j) 스핀락은 \_\_\_\_\_하면서 CPU를 소모하지만 \_\_\_\_\_이(가) 필요 없어서 락의 경쟁이 낮은 환경에서 유리하다.
- (k) 자바의 **Executor**는 \_\_\_\_\_메소드를 통해 스레드를 실행하지만, **ExecutorService**는 \_\_\_\_\_메소드를 통해 스레드를 실행한다. 차이점이 있다면 \_\_\_\_\_는 **Future** 오브젝트를 통해 결과 값을 리턴한다.
- (l) Divide-by-zero나 illegal memory access와 같이 스레드 내부에서 발생한 시그널을 \_\_\_\_\_(이)라고 한다.
- (m) 대화형 프로그래밍 환경에서는 응답시간의 평균을 줄이는 것보다 \_\_\_\_\_을(를) 줄이는 것이 더 중요하다.
- (n) 프로세스의 수가 늘어남에 따라 RMS (Rate Monotonic Scheduling) 알고리즘이 얻을 수 있는 CPU 사용률은 최대 \_\_\_\_\_%에 접근한다.

- (o) 리눅스의 `fork()`와 `pthread_create()` 함수는 둘 다 내부적으로는 커널의 \_\_\_\_\_ 함수를 호출한다.
- (p) 종료나 입출력 자원을 기다리는 상태가 아니면 CPU를 놓지 않는 방식을 \_\_\_\_\_(이)라고 부른다.
- (q) 모바일 시스템에서는 에너지를 절약하기 위해 동일한 기능을 하지만 성능이 떨어지는 코어를 추가로 가질 수 있다. 이러한 다중처리 시스템을 \_\_\_\_\_(이)라고 한다.
- (r) CPU 사용률을 이론적이긴 하지만 100%까지 올릴 수 있는 실시간 스케줄링 알고리즘은 \_\_\_\_\_이다.
- (s) 하드웨어 스레드는 CPU가 메모리에서 데이터를 읽거나 쓸 때 발생하는 \_\_\_\_\_ 현상을 활용하는 기술로 각 스레드는 별도의 \_\_\_\_\_을(를) 가지고 있다.
- (t) 프로세스의 80%가 타임퀀텀 안에 CPU 버스트를 마칠 수 있으면 평균 \_\_\_\_\_을(를) 줄일 수 있다는 \_\_\_\_\_이(가) 있다.
2. [5 pts] 아래 프로그램은 무작위 정수를 화면에 연속해서 출력한다. 이 프로그램을 참조해서 부모 프로세스가 무작위 정수를 생성하여 자식 프로세스에게 하나씩 넘겨주면 자식 프로세스는 그 수가 홀수인지 짝수인지를 화면에 출력하는 프로그램을 리눅스 `pipe`를 사용하여 작성하라. 작성된 프로그램은 컴파일 오류가 없어야 한다. 다만 시스템 호출시 발생할 수 있는 런타임 오류 검사는 생략해도 무방하다.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    srand(20200507);
    while (1)
        printf("%d\n", rand());
    return 0;
}
```

3. [5 pts]  $N$ 이 짝수인  $N \times N$  정방행렬  $A$ 와  $B$ 의 합인  $C = A + B$ 를 4개의 POSIX 스레드에서 병렬로 계산하려고 한다. 이를 위해  $A$ 와  $B$  행렬을 각각 4개의  $N/2 \times N/2$  행렬로 분할한 후, 이 분할된 행렬을 4개의 스레드에 나누어서 더한다. 아래는 4개로 쪼갠  $N/2 \times N/2$  행렬을 차례로 더해서 그 결과를 출력하는 코드의 일부분이다. 이 코드를 참조하여 4개의 POSIX 스레드를 생성하고, 각 스레드가 할당된  $N/2 \times N/2$  행렬의 덧셈을 마치면 부모 스레드는 기다렸다가 그 결과를 출력하는 코드를 작성하라. 작성된 프로그램은 컴파일 오류가 없어야 한다. 다만 시스템 호출시 발생할 수 있는 런타임 오류 검사는 생략해도 무방하다.

```
#include <stdio.h>
#define N 256
...
int main()
{
    int A[N][N], B[N][N], C[N][N];
    int i, j, k;
    int r, s;
    ...
    for (k = 0; k < 4; ++k) {
```

```

        r = (k/2)*(N/2); // r = 0 or N/2
        s = (k%2)*(N/2); // s = 0 or N/2
        for (i = r; i < r+N/2; ++i)
            for (j = s; j < s+N/2; ++j)
                C[i][j] = A[i][j] + B[i][j];
    }
    for (i = 0; i < N; ++i) {
        for (j = 0; j < N; ++j)
            printf("%d ", C[i][j]);
        printf("\n");
    }
    ...
}

```

4. [10 pts] 다음은 모니터를 세마포를 사용하여 구현한 것이다. 물음에 답하라. ★표가 붙은 문제는 5점으로 계산한다.

semaphore mutex = 1;	condition x;	semaphore next = 0;
	x.sem = x.count = 0;	next_count = 0;
function F ( ... )		
{	x.wait()	x.signal()
wait(mutex);	{	{
...	x.count++;	if (x.count > 0) {
body of F	if (next_count > 0)	next_count++;
...	signal(next);	signal(x.sem);
if (next_count > 0)	else	wait(next);
signal(next);	signal(mutex);	next_count--;
else	wait(x.sem);	}
signal(mutex);	x.count--;	}
}	}	

- (a) mutex의 용도는 무엇인가?
- (b) x.sem의 용도는 무엇인가?
- (c) x.count의 용도는 무엇인가?
- (d) next의 용도는 무엇인가?
- (e) next\_count 값을 검사해서 0보다 크면 signal(next)를 호출하는 곳이 두 군데 있다. 그 이유는 무엇인가?
- (f) ★ 세마포의 wait()와 signal()의 호출은 그 균형이 맞아야 한다. 그렇지 않으면 교착상태에 빠지기도 하고 동기화가 무너지기도 한다. 위 코드에서 프로세스가 함수 F를 시작하면 wait(mutex)를 호출한다. 그러나 종료할 때는 조건에 따라 signal(mutex)를 호출하지 않을 때도 있다. 이렇게 되면 균형이 깨져서 문제가 생기지 않을까? 이 물음에 대한 자신의 생각을 150자 이상 적어라.
5. [10 pts] 어떤 개발자가 다중 코어 환경에서 사용할 세마포의 wait() 함수를 다음과 같이 구현하였다. 물음에 답하라.

```

void wait(semaphore *S)
{

```

```

Disable interrupts;
while (test_and_set(&lock))
    /* spinlock */;
S->value--;
if (S->value < 0) {
    add this process to S->list;
    lock = FALSE;
    block();
    Enable interrupts;
}
else {
    lock = FALSE;
    Enable interrupts;
}
}

```

- (a) 개발자는 `wait()` 함수를 단일 연산으로 (atomically) 구현하기 위해 인터럽트를 차단하고 스핀락까지 사용하였다. 하나만 사용해도 충분한 것 아닌가? 이 물음에 대한 자신의 생각을 150자 이상 적어라.
- (b) `wait()`를 호출한 프로세스가 CS (Critical Section)에 들어갈 수 없는 상황이면 세마포 대기목록에 자신을 넣고 `block()`을 호출하여 스스로 멈춘다. 실행중인 프로세스가 멈추면 다른 프로세스들이 정상적으로 실행될 수 있도록 인터럽트를 허용해야 하는데, 이 코드를 보면 멈춤에서 풀린 다음에 인터럽트를 허용하고 있다. 멈추기 전에 인터럽트를 허용했어야 하는 것 아닌가? 이 물음에 대한 자신의 생각을 150자 이상 적어라.
6. [10 pts] 프로세스의 도착시간, CPU 버스트, 우선순위를 나타내는 나이스 값, 나이스 값에 따른 CPU 타임퀀텀이 아래 표와 같다. CPU 스케줄링은 현재 실행 중인 프로세스가 배정된 타임퀀텀을 소모했거나 또는 I/O를 기다릴 때 이루어진다고 가정한다. 같은 조건이면 대기순으로 스케줄링한다. (주의: 풀이 과정이 없는 답은 인정하지 않는다.)

Process	Arrival	CPU Burst	Nice	Time Quantum
$P_1$	0	7	0	3
$P_2$	3	5	0	3
$P_3$	6	5	-1	4
$P_4$	9	6	1	2

- (a) 리눅스의  $O(1)$  스케줄링 방식을 사용하여 처리한다면 시간에 따른 프로세스의 실행 순서와 평균 대기시간은 어떻게 되나? 간트차트를 사용하여 프로세스의 처리 과정을 보이고, 각 프로세스의 대기시간과 이들의 평균을 구한다. 단, 나이스 값이 작을수록 우선순위가 높고, 시간이 지나도 값의 변동은 없다고 가정한다.
- (b) 리눅스의 CFS (Completely Fair Scheduler) 방식을 사용하여 처리한다면 시간에 따른 프로세스의 실행 순서와 평균 대기시간은 어떻게 되나? 단, 가상실행시간은 나이스 값이 0일때 실제 실행시간과 같고, -1이면 실제 실행시간의 0.7배, +1이면 1.5배로 계산한다. 위 문항과 마찬가지로 간트차트를 사용하여 프로세스의 처리 과정을 보이고, 각 프로세스의 대기시간과 이들의 평균을 구한다.