COSE341-02 운영체제 중간고사 (Mid-term Examination)

Department of Computer Science and Engineering
Korea University

2023-04-20

Answer guidelines

Question	Points	Earned scores
1	10	
2	40	
3	40	
4	10	
Total	100	

Question 1:

[10 pts] 다음은 OX 판별 문항들이다. 각 소문항별로, 해당 서술이 옳으면 ◯, 옳지 않으면 ✕를 표기하시오.
(각 소문항별로 정답인 경우 1점, 오답인 경우 -1점이 부여됨)

The following are true/false questions. For each question, indicate whether the statement is true ◯ or false ✕.
(For each question, you will receive 1 point if the answer is correct and -1 point if the answer is incorrect.)

(a) 한 시점에 복수의 프로그램이 동시에 CPU 코어를 점유하고 연산을 병렬 실행하는 기술을 concurrent execution 이라고 한다.
Concurrent execution refers to the technique in which multiple programs can occupy CPU cores simultaneously and perform parallel computations at the same time.

X

(b) 가장 좋은 운영체제의 정의는 운영체제가 사용되는 목적과 상황에 따라 달라진다.
The definition of the best OS varies depending on the purpose and situation in which the OS is used.

O

(c) 프로세서는 프로그램에 대한 abstraction을 의미한다. The processor refers to the abstraction of a program.

X

(d) Dynamic linking은 상대적으로 많은 수의 프로그램을 실행시킬 때, 메모리 공간 측면에서 효율적이다.
Dynamic linking is efficient in terms of memory space when running a relatively large number of programs.

O

(e) CPU-bound 프로세스들은 대부분의 동작 시간을 입출력 (IO) 연산을 완료하는 데에 사용한다.
CPU-bound processes use most of their run-time completing input/output (IO) operations

X

(f) `fork()` 시스템 호출 시 child 프로세스는 parent 프로세스와 동일한 pid를 부여받는다.
When a system call fork() is called, the child process is assigned the same pid as the parent process.

X

(g) Voluntary yield를 이용하면 CPU 연산을 무한히 수행하여 CPU를 악의적으로 독점하는 프로세스를 방지할 수 있다.
By using voluntary yield, it is possible to prevent processes from maliciously monopolizing the CPU by performing infinite CPU operations.

X

(h) System call 호출은 mode switching과 context switching을 필요로 한다.
System call invocation requires mode switching and context switching.

X

(i) 모든 프로세스는 각각 별도의 PCB 구조를 가지고 있어야만 한다. Every process must have its own separate PCB.

O

(j) 선점 기반 스케줄링 기법은 비선점 스케줄링 기법에 비해 더 좋은 CPU 활용률을 보인다.
Preemptive scheduling shows better CPU utilization compared to non-preemptive scheduling.

X

Question 2:

[40 pts] Microkernel 구조의 게스트 운영체제를 하이퍼바이저를 통해 구동하고, 게스트 운영체제 내에서 *App K*라는 사용자 응용 프로그램을 실행한다. 이 프로그램은 하드 디스크에 대한 파일 입출력 동작(block IO)을 수행한다.

A guest operating system of a microkernel structure runs through a hypervisor and executes a user application called *App K* within the guest operating system. This program performs file input/output operations (block IO) involving a hard disk.

(a) (10 pts) *App K*가 하드 디스크에 접근하는 과정에 대해 설명하시오. 하이퍼바이저 내부에서의 동작은 하이퍼바이저가 지원해주어야 하는 필수적인 동작 또는 역할 등을 중심으로 설명하시오.
Describe the procedure of how *App K* accesses the hard disk. For the internal operation of the hypervisor, describe by focusing on the essential operations or roles that the hypervisor must support.

1. 하이퍼바이저 내부의 마이크로커널에서의 동작을 올바르게 설명함 (6점)
a. App K는 block IO를 위한 시스템 콜 (read, write) 등을 실행한다. 해당 시스템 콜은 커널 내 다수 기능을 거치게 되는데, 마이크로 커널 구조에서는 커널의 기능이 "server"로 존재하며 각각의 서버는 각각의 주소 공간을 지닌다. (2점)

b. Block IO를 수행하는 경우, VFS - 세부 파일 시스템 (e.g., EXT3), 디스크에 대한 디바이스 드라이버까지 접근하게 된다. 이 때, 각각의 서비스를 호출할때는 서비스들이 서로 다른 주소 공간을 가진 독립된 프로세스이므로, 커널이 제공하는 message passing, 즉 IPC 기능을 통해 upcall을 거쳐 호출된다. (2점)

c. Device driver는 하드 디스크에 대해 IO를 처리하기 위한 동작, 즉 데이터 입출력을 할 메모리 영역 및 디스크에 command를 생성하는데, 이 경우 디스크는 하이퍼바이저가 된다. (2점)

2. 하이퍼바이저가 제공하는 기능을 올바르게 설명함 (4점)
반가상화가 아닌 전가상화 형태의 게스트 OS의 입장에서는 하이퍼바이저가 온전히 하드 디스크가 존재하는 것과 같은 virtualized resource를 에뮬레이트하여 제공한다. 따라서 게스트 OS의 Device Driver에 도착한 IO 요청은 하이퍼바이저가 전달받는다.

IO 요청을 받은 하이퍼바이저는 해당 IO 요청에 대응되는 물리 하드디스크를 찾고, 디스크 장치에 대한 실제 IO 작업을 수행할 수 있도록 한다.

더 구체적으로는, 상이한 가상머신에서 제공되는 다양한 IO 명령들을 모으고, 각각의 가상 머신이 고립된 동작 및 성능을 제공받을 수 있게 해준다.

(b) (10 pts) (a)에서 설명한 *App K* 및 파일 입출력 동작을 이번에는 하이퍼바이저가 아니라 컨테이너를 통해 구동한다고 하자. 또한 *App K*는 microkernel 형태의 게스트 운영체제가 아닌 리눅스 (monolithic kernel) 위에서 컨테이너를 통해 구동된다. 이 때, 하드 디스크에 접근하는 과정에 대해 설명하시오.
Assume that *App K* and its file I/O operations described in (a) are now running through container instead of hypervisor. In addition, *App K* is running through the container on top of Linux (monolithic kernel) rather than microkernel-based guest OS. Describe the process of accessing the hard disk in this scenario.

컨테이너는 커널을 포함한 가상머신과 달리, 커널은 host OS의 커널을 그대로 이용한다. 즉, 컨테이너는 라이브러리와 *App K*만 포함한 형태이다.

이 때, 컨테이너는 리눅스 위에 존재하는 일종의 프로세스와 유사한데 이 프로세스가 접근가능한 자원의 상한선 (by cgroup), 접근 가능한 네트워크 및 파일 시스템 영역 (by namespace) 등이 제한된다.

Block IO를 수행하면, *App K*는 시스템 콜을 호출한다. 시스템 콜 수행을 위해 mode switch가 발생하며 VFS - specific file system (e.g., EXT3) - device driver를 거쳐 실제 하드웨어에 접근하여 IO 동작을 수행한다.

이 때, 컨테이너가 다른 점은 파일 시스템을 호출할 때, 또는 장치에 접근할 때 컨테이너에게 허용된 영역 또는 자원만 쓸 수 있도록 제한하는 커널의 기능 (cgroup 또는 namespace)가 같이 동작한다는 것이다.

[채점기준]

1) 컨테이너는 커널을 포함하지 않아 host OS에서의 기능을 순차적으로 접근하여, IO가 수행됨을 설명 (5점) : 별도의 virtual machine monitor 또는 컨테이너 엔진 내부에서의 동작이 설명되면 점수 부여 불가
- VFS, Device driver의 동작 과정을 명시하지 않은 경우 (각 -1점)
- 설명이 모호한 경우 (-2.5점)

2) 컨테이너별 허용 자원을 제어하기 위한 커널 내 기능/요소/모듈 등이 있음을 설명 (5점) : cgroup, namespace 등 구체적인 명칭은 언급하지 않아도 됨

(c) (5pts) 위 (a)와 (b)에서 설명한 과정을 비교하고자 한다. 두 가지 방식에 있어, (a)에서는 *App K*가 microkernel에서 구동된다. (b)에서의 *App K*는 monolithic kernel인 리눅스를 기반으로 구동된다. 두 방식의 장점(유리한 점)을 각각 1가지 씩 기술하시오.
Compare the procedures described in (a) and (b). In (a), *App K* runs on microkernel, while in (b), on monolithic kernel-based Linux. Describe one advantage of each method (a) and (b).

1. microkernel의 장점을 아래 중 한 가지 서술 (2.5점)
a. 커널 서비스 간 독립적인 구동 및 실행이 가능하여, fault 대응 및 소프트웨어 업그레이드 등이 용이하다.
b. 커널 서비스의 간단한 시작/종료 가능하여 불필요한 서비스는 배제하고 실행 가능하여 CPU나 메모리와 같은 자원을 효과적 관리할 수 있다.
c. 문제 발생 시 버그가 발생한 서비스만 재 부팅 가능하다.
d. 각 서버별로 주소 공간이 별도로 존재하여 독립된 실행 및 검증 가능하다. (기능별 검증이 중요한 곳에서 사용하기 적합)

2. monolithic kernel의 장점을 아래 중 한 가지 서술 (2.5점)
a. 응용 프로그램과 커널 또한 같은 주소 공간에 배치되어 오버헤드가 작다. 따라서 microkernel에 비해 일반적으로 처리 속도가 빠르다.
b. 커널의 기능이 모두 하나의 주소 공간에 배치되어, 커널 기능 간 호출의 병목이 적다. 따라서 microkernel에 비해 일반적으로 처리 속도가 빠르다.

(d) (5pts) 또한, (a)와 (b)에서 설명한 과정으로부터 하이퍼바이저와 컨테이너 (런타임) 간 동작 또는 복잡성을 비교하고, 두 기술의 우위에 대해 논하시오.
Furthermore, compare the operation or complexity between the hypervisor and container (runtime) based on the procedures described in (a) and (b) and discuss the superiority of the two techniques.

[채점기준] 아래 비교 기준 중 하나 이상을 설명하면 5점 부여.

1) 동작 과정의 복잡성 비교

컨테이너는 운영체제를 포함하고 있지 않으며, 일반적인 커널이 프로세스를 처리하는 과정에 고립된 자원과 성능을 제공하기 위한 추가적인 커널 서비스를 통해 구현된다. 구체적으로 동작 과정을 살펴보면, 하이퍼바이저를 사용하면 device driver에 도달한 이후 하이퍼바이저 내부에서 또 다시 여러 가상 머신의 IO 명령 등을 모으고 스케줄링하는 과정이 필요하다. 반면, 컨테이너의 경우, device driver에 도달한 뒤 바로 하드 디스크에 접근할 수 있다. 따라서 컨테이너에서의 동작 과정 복잡성이 훨씬 적어 짧은 시간 안에 의도한 동작이 가능하다.

2) 컴퓨팅 리소스 측면 효율성 비교

뿐만 아니라 하이퍼바이저는 게스트 OS의 운영체제(커널)을 별도로 구동하므로 호스트가 되는 컴퓨터의 컴퓨팅 리소스를 더 많이 사용하게 된다. 따라서 물리적인 컴퓨팅 리소스의 효율성 측면에서도 하이퍼바이저보다 컨테이너가 유리하다.

3) 고립 및 안정성 비교

다만, 성능적으로는 하이퍼바이저가 컨테이너 보다 우위에 있더라도, 컨테이너는 가상화된 guest가 커널을 포함하고 있지 않기 때문에 일반적인 운영체제가 제공하는 하드웨어에 대한 고립 및 안정성을 제공할 수 없다.

구체적으로 살펴보면 컨테이너는 CPU, 메모리, 파일 시스템 영역 등 사전에 정의된 자원에 대한 고립만 제공된다. 즉, 동작 중 특정 컨테이너가 host OS에 문제(fault)를 발생시키는 경우 함께 동작중인 타 컨테이너들도 모두 해당 fault의 영향을 받는다는 점에서 자원 고립이나 안정성 측면에서는 그 격리성이 하이퍼바이저보다 불리하다.

(e) (10pts) 마지막으로 (d)에서 비교한 두 가상화 기술 중에서 상대적으로 성능이 낮은 가상화 기술을 개선하기 위한 방법을 2가지 제안하시오. 개선방식은 소프트웨어 접근 및 하드웨어 접근 등 어떤 방식을 활용해도 무방하다.
Suggest two ways to improve the virtualization technique that is relatively low in performance based on the comparison in (d). The improvement methods can take any approach, such as software-based or hardware-based.

[채점기준] 구체적인 디테일은 제시되지 않더라도, 소프트웨어 및 하드웨어 접근에 있어서 핵심적인 아이디어를 정확히 설명하면 점수 부여. 아래의 답안이 아니더라도, 논리적으로 합당하며 현실적으로 실현 가능하다면 점수 부여함.

한 가지 방법 당 5점 부여.

1) 소프트웨어 접근 1: 반가상화

하이퍼바이저의 주요 병목은 게스트 운영체제가 자신이 가상화되었다는 사실을 모르기 때문에, 일반적인 운영체제가 바라보는 하드웨어를 완전히 에뮬레이션해주어야 한다는 것에 있다.

이를 해결하기 위해, 반가상화(para-virtualization) 기법을 활용할 수 있다. 반가상화는 게스트 OS의 커널이 자신이 가상화되었음을 알고, 하드웨어에 대한 접근이 아니라 "하이퍼바이저"와 직접 통신하여 [게스트 OS -에뮬레이트된 하드웨어 -하이퍼바이저] 과정이 아닌 [게스트 OS -하이퍼바이저] 과정으로 자원 접근을 가능하게 한다. 이 때, 게스트 OS가 하이퍼바이저와 직접 통신하는 수정된 API를 하이퍼콜(hyper call)이라고 한다.

2) 소프트웨어 접근 2: 경량화된 게스트 OS

두 번째 방식은 VM 자체를 가볍게 만드는 것이다. 일반적인 운영체제는 다양한 하드웨어, 응용 프로그램들을 구동시키기 위해 범용적인 컴포넌트를 모두 가지고 있다. 즉, 실제 사용자가 구동하고자 하는 application 또는 하드웨어가 아닌 다른 목적의 구현도 함께 포함하고 있어 그 규모가 크고 복잡해진다.

이를 개선하기 위해, 사용자의 프로그램이 호출하는 라이브러리 및 사용하는 하드웨어의 기능 (디바이스 드라이버)만 선택적으로 컴파일 및 구성하는 기법을 생각해볼 수 있다.

이러한 방식의 운영체제를 일반적으로 라이브러리 OS라고 하며 대표적인 예로 "unikernel"이 있다.

3) 하드웨어 접근: CPU, IO 장치 등 하드웨어 장치 자체가 가상화를 알고(aware) 최적화

또한, 소프트웨어가 아니라 하드웨어가 가상화를 위한 최적화를 수행할 수 있다. 이러한 방식은 하드웨어와 가상머신이 하이퍼바이저를 거치지 않고 바로 하드웨어를 사용할 수 있는 가능성을 제공한다.

가령 INTEL VT-d는 IO 장치가 자신에게 IO 명령을 내리는 게스트가 다수 있을 수 있음을 인지하고, 하드웨어적으로 각 가상머신에 바로 부착시킬 수 있는 IO command queue 등을 제공한다. 유사한 형태로 CPU와 VM 간 직접 제어를 일부 가능하게 하는 기술로 VT-x가 있다.

Question 3:

[40 pts] 새로운 운영체제인 **KU**nux를 구현하고자 한다. **KU**nux에서 동작하는 CPU 스케줄링 방식을 설계하고자 할 때, 아래의 질문에 순차적으로 답하시오.
Suppose we implement a new operating system called **KU**nux. When designing a CPU scheduling algorithm for **KU**nux, answer the following questions.

(a) (10 pts) **KU**nux는 CPU 코어가 1개인 컴퓨터에서, 2개의 프로세스(프로세스 A와 프로세스 B)를 구동중이다. 두 프로세스의 스케줄링에 50 ms를 time slice로 하는 round robin 스케줄링을 이용한다고 하자. 현재 CPU 코어에 프로세스 A가 실행중이고 프로세스 B는 ready 상태일 때, **KU**nux가 50 ms가 지났음을 알아채고 프로세스 B를 실행시키게 되는 과정에 대해 순서대로 설명하시오.
KUnux is running two processes (process A and process B) on a computer having 1 CPU core. For scheduling two processes, **KU**nux uses a round-robin scheduling method of a time slice 50 ms. Assuming that process A is currently running on the CPU core and process B is in a ready state, describe the procedure sequentially in which the **KU**nux detects that 50 ms has passed and switches to running process B.

라운드 로빈 스케줄링은 CPU 자원을 각 프로세스에 일정 시간 할당하고, 이후에 다른 프로세스에게 CPU 자원을 넘겨줌으로써 복수의 프로세스들에 대해 CPU 자원을 효과적으로 분배할 수 있다.

CPU는 주기적으로 하드웨어 인터럽트를 발생시키는데, 이 인터럽트는 KUnux 커널에서 인터럽트 핸들러에 의해 처리된다. 프로세스 A가 CPU 자원을 할당받은 시간이 완료되고, 프로세스 B가 CPU를 사용하게 되는 과정은 다음과 같다.

1. 인터럽트 발생: 프로세스 A가 실행 중에 있을 때, 의도된 타임 슬라이스마다 타이머 인터럽트가 발생한다. 여기서는 50 ms마다 발생한다. (2점)

2. mode switching: 타이머 인터럽트가 발생하면, 프로세스 A는 모드 스위치가 발생하며 운영체제 내의 인터럽트 핸들러가 호출되고 스케줄러 루틴을 호출하게 된다. (2점)

3. 다음 프로세스 선택: 운영체제는 프로세스 스케줄링 알고리즘에 따라 다음에 실행할 프로세스를 선택한다. 라운드 로빈 스케줄링에서는 ready 큐에 있는 다음 프로세스를 순차적으로 선택한다. (2점)

4. context switching 1) 현재 실행중인 프로세스의 저장 (2점) 현재 실행중인 프로세스 A가 종료상태(terminated)가 아니고, 다음 번 타임슬라이스가 돌아올 때 실행이 지속된다면, ready상태로 바뀌고 ready큐로 인입되며, 프로세스 A의 PCB를 향후 복원을 위해 저장한다.

2) 새로운 프로세스의 로드 (2점) 그리고 다음 프로세스인 프로세스 B의 상태(from PCB)가 로드되고, 해당 프로세스가 실행된다. 이 과정을 이전 프로세스의 상태를 복원하고, 새로운 프로세스의 상태를 로드하는 컨텍스트 스위칭이라고 한다.

이를 통해 선택된 프로세스가 CPU 자원을 할당받아 실행된다.

(b) (10 pts) **KU**nux의 개발자들은 (a)에서 사용한 스케줄러의 timeslice를 5 ms로 변경하였다. 이 때, **KU**nux 스케줄러가 기존 방식 (a)에 비해 어떤 점이 개선되고, 어떤 점이 악화되는 지 1가지 씩 설명하시오.
The developers of **KU**nux now change the time slice of the scheduler used in (a) to 5 ms. Describe one improvement and one deterioration of the **KU**nux scheduler compared to the previous method (a).

CPU 스케줄러의 time silce는 프로세스가 한번 CPU 자원을 할당 받을 때 연속적으로 CPU 자원을 할당받는 시간을 의미한다.

(개선) (5점) - 프로세스가 보다 빠르게 CPU 자원을 재할당 받을 수 있으므로, 응답성이 개선된다.

(악화) (5점) - 프로세스간 문맥 교환이 더 자주 그리고 더 많이(1회 CPU 자원 할당 시간이 작으므로) 일어나기 때문에, 병목이 발생한다.

(c) (10 pts) 인공지능 학습 또는 추론을 수행하는 프로세스들을 떠올려보자. 이 프로세스들은 GPU를 일종의 CPU처럼 사용하여 연산을 수행한다. 이 때, **KU**nux의 개발자들이 GPU에 대한 스케줄링 방식을 새롭게 개발하려 한다. 스케줄링 방식은 GPU에 대해 작업들의 평균 대기시간 또는 평균 완료시간 (job completion time)을 최적화하고자 한다. 해당 목적에 적합한 스케줄링 방식을 제안하고 설명하시오.
Consider processes performing artificial intelligence training or inference. These processes use GPUs similar to CPUs for computations. Then, the developers of **KU**nux try to develop a new scheduling method for GPUs. This scheduling method aims to optimize the average waiting time or job completion time of tasks for the GPU. Suggest and explain a suitable scheduling method for this purpose.

Shortest remaining time first가 적합하다.

Ready time 또는 job completion time은 앞서 실행되고 있는 작업이 빨리 끝나면 끝날수록 개별 job의 대기 시간이 짧아지므로, 짧은 job이 먼저 실행될 수록 평균 ready time이 개선된다.

SRTF는 ready queue에 존재하는 프로세스들 중 remaining time이 가장 짧은 작업을 실행시키되, 선점형으로 실행시키므로, 다양한 스케줄링 기법 중 대기시간을 최소화할 수 있는 기법이다.

단, SRTF 알고리즘의 경우 잔여시간 (또는 예상 실행시간)이 기존에 스케줄러에 등록된 프로세스의 예상 실행시간보다 짧은 프로세스가 지속해서 대기 큐에 등록되는 경우, 해당 프로세스만 우선적으로 스케줄링하여 처리시간이 긴 프로세스가 CPU 자원을 할당받지 못하는 기아 현상이 발생할 수 있다. 따라서, 이를 방지하기 위한 기법이 필요한데, 간단하게는 프로세스 스케줄링의 시간에 따른 가중치를 부여하는 Aging 기법을 활용할 수 있다.

- SRTF 방식에 대한 제시 (4점) - SRTF가 ready 시간을 최적화하는 이유에 대한 설명 (3점) - 기아현상에 대한 언급 및 해결책 (3점) ***만약, 기아현상에 대한 언급 및 해결책이 (d)에 설명이 되어있다면 점수 부여

- 타 스케줄링 정책 언급 및 적절한 논리 제시의 경우 7점 부여

(d) (10 pts) (c)에서 제안한 스케줄링 방식을 구현하는 데에 어떤 어려움이 있을 지, 해당 어려움을 어떻게 해결할 수 있을 지 서술하시오.
Explain any difficulties in implementing the scheduling method proposed in (c) and address how the difficulties can be solved.

어려움: 각 프로세스가 GPU를 점유하는 시간, 프로세스별 remaining time (실행 시간)을 미리 알기 어렵다 (5점)

해결: 과거의 history를 기반으로 잔여 시간을 알아내거나 별도의 예측 또는 모델링을 수행할 수 있음 (5점)

[예시 답안]

그러나, SJF/SRTF 알고리즘은 작업의 예상 실행 시간이 정확하지 않은 경우 문제가 발생할 수 있고, 실제로 예상 실행 시간을 실행 없이 예측하기란 매우 어렵다. 이러한 스케줄링 알고리즘에서 예상 실행시간을 계산하기 위해 GPU Burst의 특성을 이용한 예측 등이 존재한다.

[GPU 예측 기법 예시] Yang, Gyeongsik, et al. "Prediction of the resource consumption of distributed deep learning systems." Proceedings of the ACM on Measurement and Analysis of Computing Systems 6.2 (2022): 1-25.

[GPU Burst 계산 알고리즘 예시]

$$t_n = \ actual\ length\ of\ n^{th}\ GPU\ burst \tag{1}$$

$$\tau_{n+1} = predicted\ value\ for\ the\ next\ GPU\ burst \tag{2}$$

$$\alpha, 0 \leq \alpha \leq 1 \tag{3}$$

$$Define : \tau_{n=1} = \alpha \times t_n + (1-\alpha) \times \tau_n \tag{4}$$

$$(\alpha\, weight,\ normally\ 0.5) \tag{5}$$

Question 4:

[10 pts] 3번에서의 **KU**nux를 다음과 같이 새로운 방식으로 설계한다고 하자. 수업시간에 논의된 내용 하에서, **KU**nux의 방식이 커널 및 사용자에게 어떠한 영향을 가져올 지 서술하시오.
팁: 문항에서 언급하는 운영체제의 abstraction 또는 개념이 일반적인 운영체제에서 필요한 이유 (목적) 및 그 역할을 떠올려보시오.

Suppose we redesign **KU**nux of Question 3. Based on the discussions in class, describe how **KU**nux's approach would affect the kernel and the user.
Tip: Consider the reason (purpose) why the operating system's abstraction or concepts, mentioned in the question, are needed in the general operating system and its role.

(a) (4 pts) **KU**nux에서는 일반적인 운영체제와 달리, 프로세스라는 abstraction이 사라진다고 하자. 이 때, 발생할 수 있는 문제점을 1가지 서술하시오.
Assume that in **KU**nux, the idea of process abstraction does not exist, unlike in a typical OS. Describe one potential problem that could arise as a result.

[예시 답안]

프로세스 개념이 운영체제에서 사라지면 프로그램이 실행중인 상태에 대해 관리 및 제어가 어려워진다. 구체적으로는 다음과 같다.

- 프로그램이 실행중인 상태에 대해 관리 및 제어하기 어려워짐.

- 하나의 프로그램이 여러 실행 상태를 가질 때, 각각을 구분할 수 없음.

- CPU를 점유하고 있는 하나의 프로그램이 다른 프로그램으로 교체될 수 없으며, 교체를 위해서는 컨텍스트 스위칭, 즉 각 프로그램별 상태와 PC 등을 저장하기 위한 새로운 메커니즘이 필요함.

- (resource isolation) 프로세스 개념은 프로세스 간에 자원을 분리하고 보호하기 위한 중요한 메커니즘이다. 프로세스 개념이 없어지면, 프로세스간 자원 분리가 어려워져 한 프로세스가 다른 프로세스의 자원에 접근하는 등 보안과 실행 안정성 부분에서 문제가 발생할 수 있다.

- (scheduilng) 프로세스 개념은 각각의 프로세스를 분리하여 스케줄링하는 것을 가능하게 하는 중요한 개념이다. 프로세스 개념이 없으면, 동시에 실행되는 프로그램들이 보다 저수준에서 CPU 등의 자원을 할당받기 위해 경쟁하므로, 시스템의 전체적인 성능이 저하된다. 또한, 컨텍스트 스위칭을 위해서 각 프로그램별 상태와 프로그램 카운터 등을 저장하기 위한 새로운 메커니즘을 개발해야 한다.

- (debugging) 프로세스 개념은 시스템에 문제가 발생했을 때 프로세스 별로 상태를 추적하고 디버그하기 쉽게 만든다. 프로세스 개념이 없으면, 이 과정이 어려워진다.

- (security) 프로세스 개념은 각각의 프로세스가 독립적으로 실행되도록 보장한다. 프로세스 개념이 없으면 프로세스 간의 데이터 공유나 보안 이슈가 발생할 수 있.

- 예시 답안 외에도 발생할 수 있는 문제점과 그에 대한 타당한 이유를 낸다면 OK

- (문제점에 대한 설명이 부적절한 경우 - 2점 감점)

(b) (6 pts) **KU**nux가 프로세스라는 abstraction을 가지고 있지만, protection domain을 제공하지 못한다고 하자. 이 때, 해당 방식의 이점을 inter-process communication의 관점에서 설명하시오.
Assume that **KU**nux has process abstraction but does not provide the protection domain. Explain the advantages of this approach from the perspective of inter-process communication.

[예시 답안] (6점. 단, 논리적으로 말이 되는 이점을 설명한 경우 점수 부여) Protection domain 하에서 프로세스 통신을 가능하게 하기 위해서는 IPC라고 하는 별도의 커널이 제공하는 기법을 활용해야만 함.

가령 각 프로세스의 주소 공간에 별도로 공유되는 메모리 영역을 맵핑해둔다던가, 별도로 커널의 공간을 거쳐가는 메시지 교환 방식을 사용해야만 함.

protection domain이 제공되지 않는다면, 이러한 별도의 방식 없이 직접 프로세스들이 통신을 수행할 수 있음.

즉, 프로세스간 통신에 커널이 제공하는 IPC 메커니즘을 통하지 않고, 직접적으로 메모리에 접근할 수 있으므로 IPC 동작에 발생하는 병목 현상이 줄어들게 된다. 즉, 데이터 공유에 걸리는 지연 또는 데이터 공유를 위해 커널이 사용하는 CPU 사용량 등이 감소한다.

note) Protection Domain은 각 프로세스에 할당된 메모리 영역을 의미하며, 허가되지 않은 메모리 영역의 값 수정이나 접근으로 부터 프로세스를 보호하는 것이 목적이다. Protection Domain이 운영체제에서 사라지면, 이런 보호가 불가능해진다.

(전체적인 맥락은 유사하나 잘못된 설명이 있을 경우 - 2점 감점)
(일부 맥락이 유사하나 IPC 관점에서의 설명이 불충분한 경우 - 4점 감점)

중간고사 시점까지 고생 많으셨습니다. 남은 기간 또한 힘내서 열심히 해봅시다.