2021년 2학기 운영체제 CSE30100

중간고사 대체리포트

학번	2020105695
학과	소프트웨어융합학과
이름	김희성

1. OS(Operating System)란?

key word : OS, Memory Hierarch, Interrupt, Memory, CPU, von neumann

- 운영체제 : 사용자가 안전하고 편하게 컴퓨터를 사용할 수 있도록, 하드웨어 장치를 효과적으로 관리해주는 프로그램.

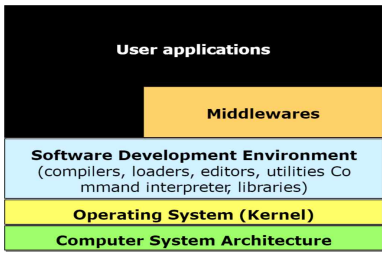
- **운영체제 구성요소** : middle ware, kernel, system program

kernel : OS에서 항상 작동하는 프로그램 ex) scheduler, 보안

system program : OS가 하는 일중에서 사용자가 사용하기 쉽게 뽑아 놓은 것, 혹은 OS에게 보내는 요청 ex) 파일 탐색기

middle ware : 응용프로그램 개발을 위해, OS에 추가적인 서비스를 제공 ex) DBMS, JVM

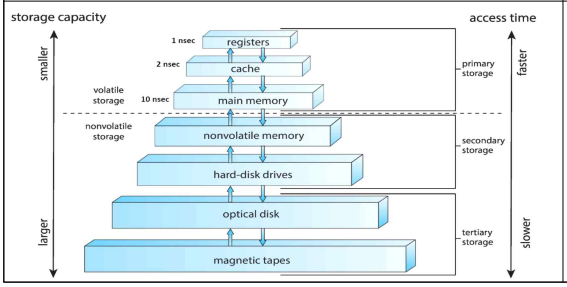
운영체제 구성요소



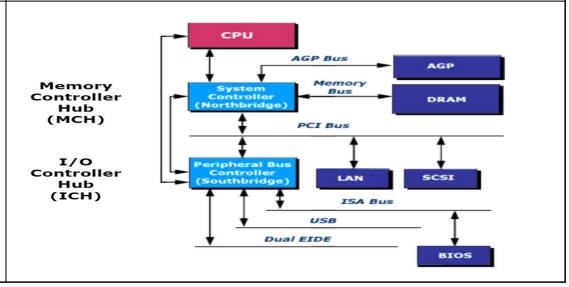
- **HW 관리** : memory, IO장치, cpu 등의 자원의 독점을 막음.

1) memory

메모리 계층화



현대 컴퓨터 구조



메모리 계층화를 통해, 속도가 빠르지만 비싼 부품을 효과적으로 사용. 아래로 갈수록 큰 데이터가 느리게 움직임. 위로 올라갈수록 적은 데이터가 빠르게 움직임. 컴퓨터 구조에서 이런 데이터들은 System Bus를 통해서 움직임.

2) interrupt

interrupt : 외부의 HW 장치에서 IO가 들어왔음을 알리는 방법 ex) pooling vs Interrupt

pooling 방식 : OS가 수시로 I/O buffer를 확인. (거의 쓰지 않음.)

Interrupt 방식 : 장치가 I/O를 끝낸 후, CPU에서 끝난다는 명령어를 줌. (현대 쓰는 방식)

trap : 장치 내부적으로 일으키는 interrupt. ex) debugging시의 break point

★ I/O 처리 순서 ★

1. 인터럽트 발생
2. cpu는 usermode에서 하던 일을 멈추고(PCB 저장), kernel mode로 진입
3. vector table에서 적절한 인터럽트 벡터를 fetch
4. interupt routine을 수행
5. 다시 user mode로 돌아와서 하던 일을 계속함

3) CPU
컴퓨터는 폰 노이만 구조에 따라서 1. fetch 2. decode 3. execute 3단계에 걸쳐 진행.
fetch : auxiliary memory에서 main memory로 가져옴.
decode : 실행해야 하는 명령어가 무엇인지 파악함. (cpu의 control unit)
execute : 계산을 수행함. (cpu의 arithmetic logic unit)
하버드 방식은 3가지 과정에서 data와 instruction을 분리함. 명령과 데이터를 따로 register에 받음.
다양한 프로그램들이 올라가서 scheduling을 통해 cpu를 독점적으로 사용하는 것을 막음.
ex) time interrupt를 통해 다른 program도 골고루 실행 될 수 있도록 함.

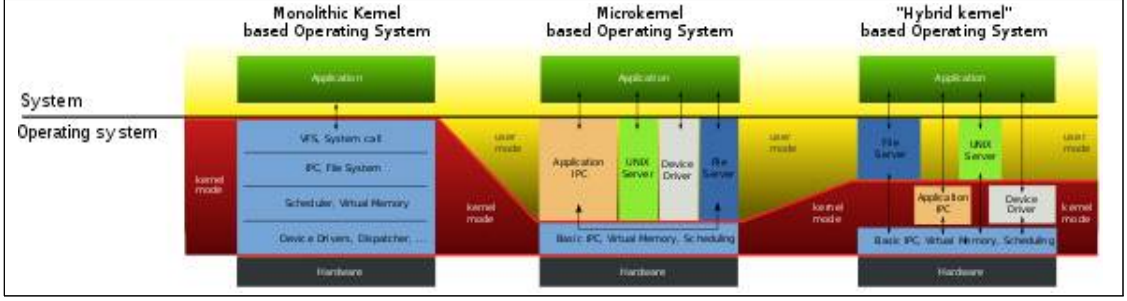
- 기타
이밖에도 보안, HW 장치들의 추상화, 네트워크, shell script CLI compile 등등의 일을 함.

2. System Structure
key word : Dual mode, 종류
- Dual Mode Operation
user mode vs kernel mode
user mode : 사용자가 쓰는 application이 작동되는 mode
kernel mode : 운영체제가 사용하는 mode
user mode -> kernel mode : system call ex) posix system에서 fork로 process 만든다.
privilege instruction : kernel mode에서만 사용 가능한 명령어이다. ex) system call
나누는 이유 : user가 잘못된 방식으로 kernel 모드에 접근하면 컴퓨터가 망가지기 때문!

- 역사
main frame system: 한 가지 program만 올라가서 실행됨.
multi programmed batch system: ram에 여러 process 모아두고, I/O 할 때만 다른 process를 실행. (Monitor라는 OS가 존재했다.)
time sharing: time quentum마다 다른 process를 실행해, 여러 유저가 동시에 사용할 수 있도록 만듬. fairness를 확보함.

- 종류
Desktop System : 현재의 데스크 탑 컴퓨터의 시스템
Parallel System (concurrent: cpu 1개 vs parallel: cpu 多개) : 여러 CPU를 두어 병렬처리 시스템
Distributed System : network를 이용해 분산으로 처리하는 시스템
Embedded System : 한정된 ram과 processor로 만들어야 하는 곳에 사용되는 컴퓨터 시스템
Realtime System : dead line까지 반드시 마쳐야 하는 문제에서 쓰이는 시스템

- Micro Kernel vs Monolithic Kernel



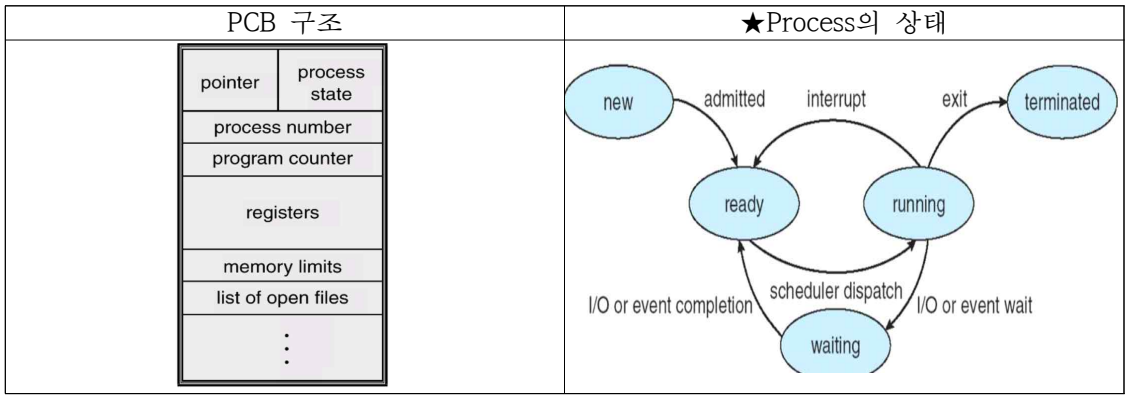
kernel의 이식성과 유연성을 위해, Micro kernel을 사용.
좋은 성능을 위해, monolithic kernel을 사용.

3. Process와 Thread

key word : PCB, State, IPC

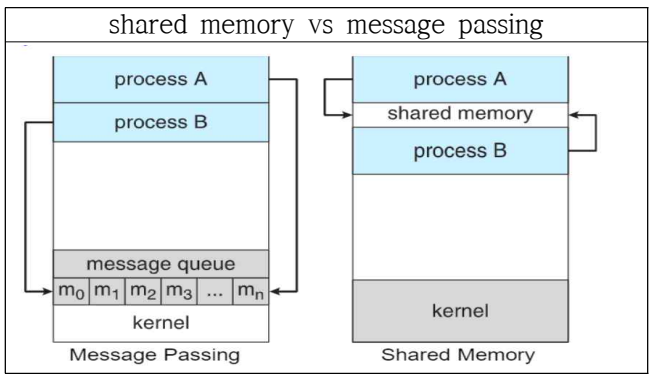
Process : 현재 main memory에서 실행되고 있는 프로그램과 register에 존재하는 값들.
Thread : scheduling의 최소 단위로, process안에서 독자적인 stack을 갖으며 실행되는 것. process 안에 여러 thread가 들어가서 실행 됨.
process와 thread의 개수 비에 따라, 1. Many-to-One 2. One-to-One 3. Many-to-Many으로 나뉨

- PCB(Process Control Block): 현재 process의 상태를 저장하기 위한 자료구조



- IPC (Inter Process Communication)

shared memory : kernel을 이용하지 않고, 공유된 메모리 공간에 정보를 넘겨 줌.
overhead가 적지만, synchronization issue가 일어날 수 있음.
message passing : kernel을 이용해서 정보를 넘겨 줌.
overhead가 크나, synchronization issue가 적음. 안정적임.



Direct Communication : 받을 process를 지정함. 한 쌍의 process들 끼리만 가능
Indirect Communication : 받을 process를 지정하지 않음.
mailbox에 넣어 둠. 동일한 mailbox를 갖으면 볼 수 있음.

- Process vs Thread

process : 독립적인 메모리 공간을 갖음. ex) fork()
Thread : 독립적인 stack(함수), 와 레지스터(pc, sp)을 갖고, data와 code section을 공유. ex) pthread()

4. Scheduling

key word : Time Quantum, Context Switching, Long term, Sort term, FIFO, SJF, Priority queue, RR, EDF Preemptive, Non Preemptive, Starvation

- Scheduling 정의

컴퓨터에 여러 가지 process를 동시에 실행할 때, 어떤 process와 Context Switching[1] 할지 결정하는 것.

- Scheduling 하는 이유

process는 'I/O 처리'와 'cpu 처리' 두 가지 처리가 필요함. 하지만 process가 I/O를 하는 동안에는 cpu 처리를 받을 수 없음. cpu의 처리속도는 I/O보다 빠르기 때문에, IO 시간에 다른 process의 cpu 처리가 진행 돼야 함.

많은 process들의 average waiting time을 맞추기 위해서 scheduling 돼야 함. 하지만 context switching할 때 overhead가 있기 때문에 적절한 time quantum을 맞춰야 함.

- scheduling 기준

CPU utilization && Through put : 늘리기
turn around time(끝나는 시간) && waiting time && Response time(반응시간) : 줄이기
fairness && balanced
starvation[2] 방지

- cpu bounded process vs IO bounded process

cpu bounded process : 슈퍼컴퓨터에서의 기상예측같이, 복잡한 연산이 필요한 process
I/O bounded process : game 같이, 반응성이 좋아야 하는 process

- Long Term Scheduling vs Short Term Scheduling

Long Term Scheduling : Auxiliary Memory에서 => Main Memory로 이동
Short Term Scheduling : Main Memory에서 => CPU로 이동

- FIFO (First In First Out) Queue

처음 들어온 process가 끝날 때 까지 다른 process는 진행되지 않음. 최악의 average waiting time을 보임.

- SJF (Shortest Job First)

가장 이상적인 방법. time quantum마다 남은 burst time이 가장 적은 process부터 수행. 하지만 남은 burst time을 정확히 예측하는 것이 불가능하기 때문에 쓰이지 않음.

- Preemptive (SRJF, Sortest Remaining Job First) vs Non Preemptive

Preemptive : "새로 들어온 process의 남은 burst time"이 "cpu에서 작업하고 있는 process의 남은 burst time"보다 작을 때, 새치기를 할 수 있는 경우이다.
Non Preemptive : "새로 들어온 process의 남은 burst time"이 "cpu에 작업하고 있는 process의 남은 burst time"보다 작아도, 현재 진행 하고 있는 process부터 끝냄.

---

1) cpu에서 처리되고 있는 process와, 기다리고 있는 process의 상태를 바꾸는 것
2) 한 프로세스가 자원을 독점해서, 다른 process에 자원이 할당되지 않는 것

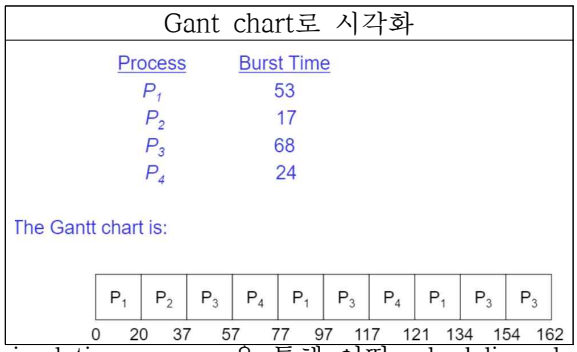
- Priority Queue
process에 우선순위를 부여해, 우선순위를 기준으로 cpu에게 처리되는 알고리즘.

- RR (Round Robin)
priority 같은 process는 한번 씩 실행함. context switching을 통해 priority가 낮은 process도 cpu의 처리를 받을 수 있음. 이를 통해 fairness를 맞춤.

- EDF(Earliest Deadline First)
realtime system에서 쓰이는 scheduling 기법. dead line에 가장 가까운 process를 먼저 처리.

- Multi Level Queue vs Multi Level Feedback Queue
위에서 언급한 다양한 알고리즘을 섞어서, 여러 가지 queue로 구성. feedback의 의미는 다른 queue로 옮겨갈 수 있다는 것.
ex) aging 기법으로, FIFO에서 시간이 지나면 realtime queue에서 실행.
ex) IO가 많으면, round robin queue에서 FIFO로 옮겨갈 수 있다.

- 기타



simulation program을 통해 어떤 scheduling algorithm이 좋은 성능을 내는지 알 수 있음.

5. Synchronization

key word : Critical Section, Race condition, Peterson Algorithm, Atomic Instruction, Mutex, Semaphore, Monitor, Disabling Interrupt

- Race Condition
서로 다른 process가 shared memory 공간에 동시에 접근해서 해당 값을 변화시키거나 참조하려는 경우. (process가 아닌 thread 일 수 있음.)
race condition에서, synchronization을 하지 않으면, 결과 값이 불규칙적으로 나올 수 있음.
∵ 동시에 접근 할 때, context switching이 일어나면 값이 바뀌기도 전에 덮어 써질 수 있음.

- ★Critical Section[3]에서 race condition을 피하기 위한 조건
1. mutually exclusive : critical section 안에는 한 가지 process나 thread만 접근 돼야 함.
2. progress : critical section 안에 아무도 없으면, 해당 process나 thread가 진행 돼야 함.
3. bounded waiting : 기다리는 프로세스는 적정한 시간이 지난 후, 실행돼야 함. 다시 말해, starvation이 일어나면 안 됨.

---
3) code상에서 race condition이 일어나는 부분

- SW적 해결 방법

Peterson Algorithm	Bakery Algorithm

do {
flag [i]:= true;
turn = j;
while (flag [j] and turn = j) ;
critical section
flag [i] = false;
remainder section
} while (1);
	do {
	choosing[i] = true;
	number[i] = max(number[0], number[1], …, number [n – 1])+1;
	choosing[i] = false;
	for (j = 0; j < n; j++) {
	while (choosing[j]) ;
	while ((number[j] != 0) && ((number[j],j) < (number[i],i)))
	}
	critical section
	number[i] = 0;
	remainder section
	} while (1);

Peterson Algorithm : boolean flag(bool array)와 turn을 이용해서 '두 process' 간의 critical section의 동시 접근을 막음.

Bakery Algorithm : ticket과 pid를 통해 '여러 process' 간의 critical section 동시 접근을 막음.

>> 총평 : 둘 다 busy waiting[4] 때문에 성능이 저하 됨. overhead가 큼. 이 방법을 Lock spin lock 이라고도 함

- HW적 해결 방법

Atomic Instruction : 해당 함수가 진행 될 동안 context switching이 발생하지 않음.

test and set	swap

boolean TestAndSet(boolean &target) {
boolean rv = target;
target = true;
 	
return rv;
}
	void Swap(boolean &a, boolean &b) {
	boolean temp = a;
	a = b;
	b = temp;
	}

test and set : target을 true로 만들어서 lock을 건다. 그래서 mutual exclusive를 구현.

swap : key 값을 통해 lock을 건다. mutual exclusive를 구현.

> bounded waiting을 달성 하지 못함. waiting 하다가 random한 값이 나가기 때문. 어떤 process는 starvation 상태가 될 수 있음.

Disabling Interrupt : OS가 수시로 context switching을 막는다. overhead가 커서 잘 쓰지 않는다.

- Higher-Level Synchronization

semaphore: critical section에 들어갈 수 있는 process의 개수를 제한해, race condition을 막음. 기다리는 process 들이 busy waitting 하지 않고, cpu를 쓰지 않는 queue에서 대기함.

binary semaphore(mutex) : int의 값이 1일 때.

counting semaphore : int가 1 이상일 때.

ex) wait(s) signal(s)

이를 통해서 process를 순서대로 진행시킬 수 도 있음.

---

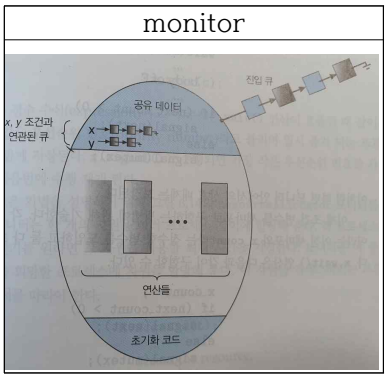
4) wait 하는 중에, 다른 process가 끝날 때 까지 cpu를 쓰면서 기다리는 연산 수행. waiting 하면서 실행이 끝났는지 계속 확인 함. splin lock이라고도 불림

monitor && critical region : 언어 차원에서 지원해 줘야함.

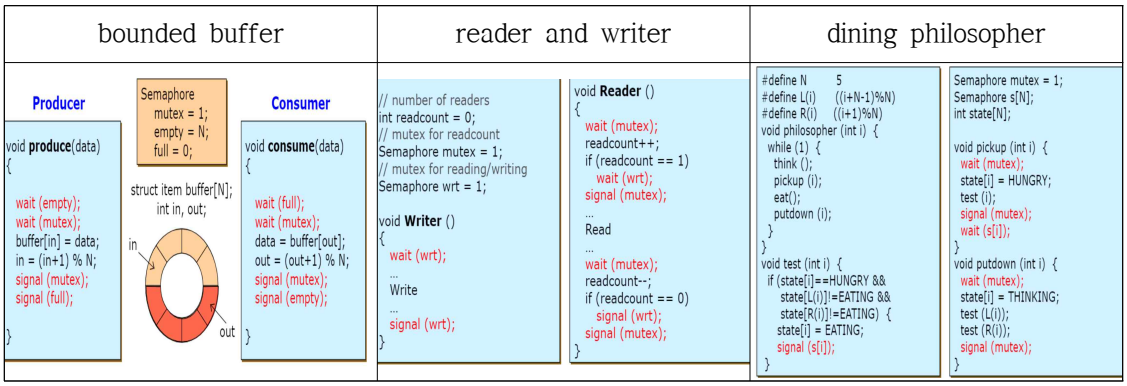
monitor : 모니터 안에는 한 가지 프로세스만 들어가도록 언어가 지원.

critical region : 변수를 공유해, 내부의 구문만 접근하도록 하여서, 효과적으로 동기화.

ex) java ‘synchronized’



- classical synchronization problem



세 가지 문제를 올바르게 synchronization하여 사용. deadlock을 피해야 함.

참고 문헌.

operating system - abraham silberschatz 10th edition