4. Process Mangement

5. CPU Scheduling

2023.06.20 **박영서 김하림 임수형**

4. Process Management

01 프로세스의 생성

- 부모 프로세스가 자식 프로세스를 생성함
- 프로세스 생성 시 트리(계층 구조) 형성
- 프로세스는 자원을 필요로 함 → 자원은 OS로부터 받음

- 자원의 공유에 따른 분류
  - 부모와 자식이 모든 자원을 공유하는 모델
  - 일부를 공유하는 모델
  - 전혀 공유하지 않는 모델: 원칙적으로는 서로 다른 프로세스이므로 자원을 부모와 공유하지 않음

- execution에 따른 분류
  - 부모와 자식이 공존하며 수행되는 모델
  - 자식이 종료(terminate)될 때까지 부모가 기다리는(wait) 모델

01 프로세스의 생성 과정

- UNIX의 예

1. **fork()**: 복제단계
   - fork()가 새로운 프로세스를 생성
   - 부모를 그대로 복사(OS data except PID + binary) → context 복사
   - address space 할당

2. **exec()**: 새로운 프로그램을 덮어씌우는 단계
   - fork() 다음에 이어지는 exec()을 통해 새로운 프로그램을 메모리에 올림

- Copy-On-Write(COW)
  - write 발생 전까지는 부모와 자원 공유
  - write 발생 시 부모 프로세스 복제 후 새로운 프로그램을 올림

01 프로세스의 종료

1. 자발적 종료: 프로세스가 마지막 명령을 수행한 후 OS에게 이를 알려줌 **exit()**
   - 자식이 부모에게 Output data를 보냄(via wait())
   - 프로세스의 각종 자원들이 OS에게 반납됨

2. 강제 종료:부모 프로세스가 자식의 수행을 종료시킴 **abort()**
   - 자식이 할당 자원의 한계치를 넘어섬
   - 자식에게 할당시킬 task가 없거나 더 이상 필요하지 않음
   - 부모 프로세스가 종료되는 경우
   - OS는부모 프로세스가 exit하는 경우 자식이 더 이상 수행되도록 두지 않음
   - 단계적인 종료

01 PID란?

- PID = Process ID
- 운영체제에서 프로세스를 식별하기 위해 부여하는 번호를 의미

- PID 의 최대값: 32768
- 32768 인 이유는 16bit signed integer 를 사용하기 때문

- PID 의 ID 할당 방식
  - 최근 할당된 PID 에 1을 더한 값으로 할당
  - 순서대로 1씩 할당되다가 32768 을 넘어가면 다시 1부터 시작

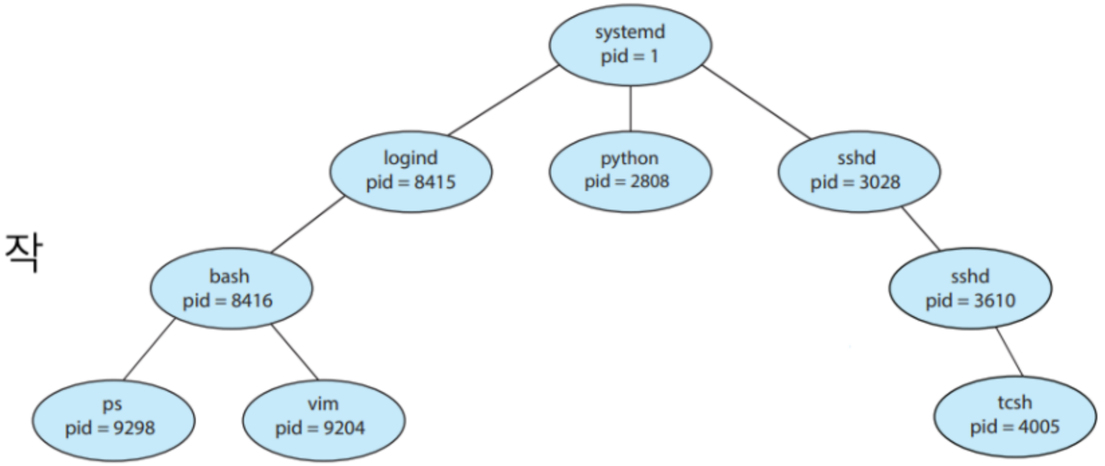


Figure 3.7 A tree of processes on a typical Linux system.

02 프로세스와 관련된 System Call

1. fork() : 부모 프로세스로부터 자식 프로세스를 생성한다.

```
#include <sys/types.h>
#include <unistd.h>

pid_t fork(void); //pid_t는 int값임

                                    //Returns: 0 to child, PID of child to parent, -1 on error
```

02 프로세스와 관련된 System Call

2. exec() : 다른 프로그램을 현재 프로세스에 로드하고 실행한다.

```
#include <unistd.h>
int execve(const char *filename, const char *argv[], const char *envp[]);

                                                    //Does not return if OK; returns -1 on error
```

위 execve() 함수 뿐만이 아니라 많은 exec() 계열 함수들이 있음

02 프로세스와 관련된 System Call

3. wait() : 자신의 모든 자식 프로세스를 기다림. waitpid(-1, &wstatus, 0)과 동일. 자세한 내용은 man 참고

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait(int *statusp); // 포인터를 넣어서 자식 프로세스가 종료될 때 어떤 상태였는지를 확인할 수 있게 함

                                                    //Returns: PID of child if OK or -1 on error
```

하림 : wait() 시스템 콜이 존재하는 이유는 뭔가요? wait() 시스템 콜의 목적에 대해 설명해주세요

1) 자식 프로세스가 종료되기 전에 부모 프로세스가 종료되는 일을 막음
2) 자식 프로세스의 종료 상태에 따라 부모 프로세스에서 처리
3) 동시성을 제어하는 데에도 사용할 수 있음

02 프로세스와 관련된 System Call

4. exit() : 현재 프로세스를 정상적으로 종료
5. abort() : 현재 프로세스를 비정상적으로 종료
   : 수업에서는 abort를 부모 프로세스에서 자식 프로세스를 종료시키는 것이라 했지만
   사실은 현재 프로세스에서 SIGABRT 시그널을 발생시키고 해당 시그널을 받은 커널이 종료시키는 것

```
#include <stdlib.h>

void exit(int status); //종료될 때 부모 프로세스에 알려줄 자신의 status를 parameter로 가짐
void abort(void);      //SIGABRT 시그널을 발생
```

03 Interprocess Communication(IPC)

독립적 프로세스(Independent Process)

- 프로세스는 각자의 주소 공간을 가지고 수행되므로 원칙적으로 한 프로세스는 다른 프로세스의 수행에 영향을 미치지 못한다.

협력 프로세스(Cooperating Process)

- 하지만 프로세스 협력 메커니즘을 통해 한 프로세스가 다른 프로세스의 수행에 영향을 미치게 만들 수 있다.

03 Interprocess Communication(IPC)

프로세스 간 협력 메커니즘(IPC, Interprocess Communication)

프로세스 간의 협력을 가능케 하는 데에는 여러 방법이 있다.

메시지 전달 방법(message passing)

process 사이에 공유 변수를 일체 사용하지 않고 서로 메시지를 주고 받음으로써 협력함

하지만 process들이 서로 독립적이므로 커널을 통해서 메시지를 전달해야 함

하림 : message passing에서 사용되는 message는 정확히 뭔가요?어떤 형태를 띠는지 궁금합니다

A) Message는 두 프로세스에서 주고 받을 데이터로, 구조체의 형태를 가지고 있음.
이 구조체를 양쪽에 선언해놓고 사용함.

Direct Communication : 통신하려는 프로세스의 이름을 명시적으로 표시하고 메시지를 전달

Indirect Communication : mailbox 또는 port를 통해 메시지를 간접적으로 전달. 누가 받을지는 명시하지 않음.

03 Interprocess Communication(IPC)

주소 공간을 공유하는 방법 (shared memory)

서로 다른 프로세스 간에도 일부 주소 공간을 공유하게 하는 shared memory 메커니즘이 있다.

물리적 메모리에 매핑될 때 일부 주소 공간을 공유하도록 매핑한다.

처음 주소 공간을 공유할 때에는 kernel의 도움을 받지만, 이후 협력에서는 kernel을 거치지 않고서도 가능해진다.

제한 사항

두 프로세스가 동시에 한 주소 공간에 write하는 경우가 없어야 한다 (충돌 가능성 있음)

이외에도 IPC 방법은 여러 가지가 있다. (ex. UNIX pipe)

03 Interprocess Communication(IPC)

UNIX pipe

프로세스간 통신을 위한 단방향 데이터 채널
가장 오래되고 가장 간단한 IPC 메커니즘
부모-자식 관계의 프로세스 사이의 데이터 교환을 가능하게 함

부모 프로세스에서 파이프를 하나 만든다. (int pipe(int fd[2]);)
이 파이프는 양 끝을 가지는데, 한 쪽(fd[1])은 쓰기용, 한 쪽은 읽기용(fd[0])이다.
이 fd는 file descriptor를 의미한다.

fork()를 통해 만들어진 자식 프로세스는 부모와 같은 file descriptor를 가진다.(복제)

부모가 쓰고 자식이 읽는 경우, 부모는 읽기 파이프를 닫고, 자식은 쓰기 파이프를 닫는다.
부모가 쓰기 파이프에 쓰면 자식은 읽기 파이프로 그 내용을 읽을 수 있게 된다.

5. CPU Scheduling

5.1 CPU & I/O Bursts

Bursts :
continued actions, 한 번에 전송되는 data block, Period

프로세스 관점.
CPU Burst - CPU를 사용할 때
I/O Burst - 입출력 대기할 때

CPU-I/O Burst Cycle이 존재

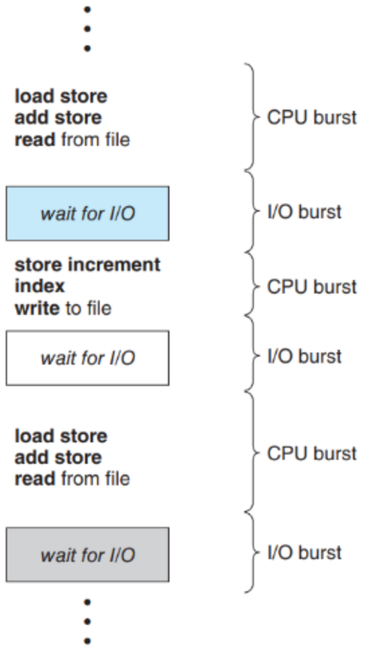


Figure 5.1 Alternating sequence of CPU and I/O bursts.

입출력이 많아서 interative한 i/o bound job의 경우 짧은 CPU 버스트가 많고, CPU bound job의 경우 CPU 버스트가 길다.

스케줄링이 없다면,
CPU가 유휴상태(idle)로 비생산적인 운용.

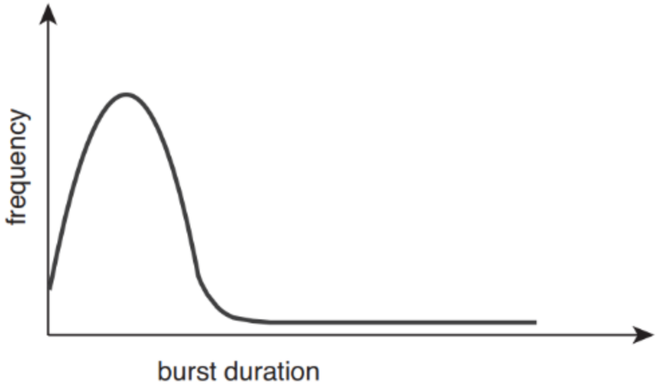


Figure 5.2 Histogram of CPU-burst durations.

운영체제는 Ready Queue에서 프로세스를 하나 고르고 실행 해야한다. -> CPU Scheduler의 역할

5.2 CPU Scheduler

Preemptive Scheduling - 하나의 프로세스가 다른 프로세스 대신에 CPU를 차지할 수 있다.
Nonpreemptive Scheduling - 하나의 프로세스가 끝나지 않으면 다른 프로세스는 CPU를 사용할 수 없다.

CPU Scheduling은 다음 4가지 상황에서 생긴다.

1. Running -> Blocked(Waiting)
   ex) I/O 요청,
   자식 프로세스 종료를 위한 wait() 호출
2. Running -> Ready
   ex) interrupt 발생시
3. Blcoked(Waiting) -> Ready
   ex) I/O 완료
4. Terminate

1, 4의 경우 Nonpreemptive, Cooperative한 스케줄링 스키마
2, 3의 경우 Preemptive한 스케줄링 스키마.

1, 4는 바로 Ready Queue의 새로운 프로세스를 실행시키므로 스케줄링 할 게 없다. 하지만 2, 3은 선택할 수가 있다. 선점적으로 선택하는 상황.

Windows, macOs, Linux, UNIX 등 거의 모든 최신 운영체제는 Preemptive Scheduling을 사용함.

5.3 Dispatcher

CPU Scheduler나 Dispatcher 둘다 운영체제의 커널 코드
CPU Scheduler가 CPU를 줄 프로세스를 골랐다면, Dispatcher가 준다.
즉, CPU core의 제어권을 넘겨주는 역할.

기능

- Context Switching
- User mode로 Switching
- 적절 시점에 프로세스를 재개하는 것.

context switching에 소요되는 시간을 dispatch latency라고한다.
가능한한 빨라야한다.

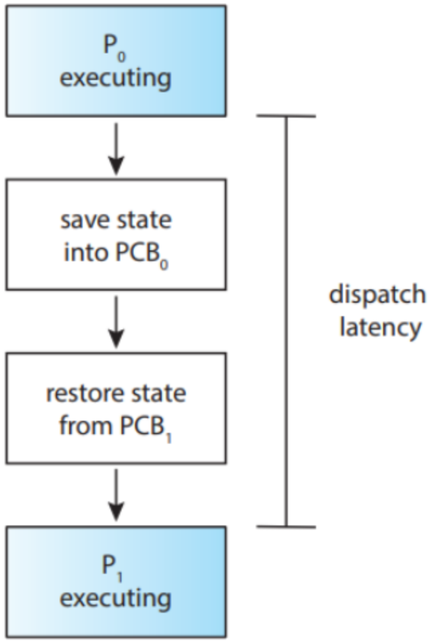


Figure 5.3 The role of the dispatcher.

```
suhyeng@DESKTOP-PL8UPTQ:~$ vmstat 1 3
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0      0 26029608   8356  76400    0    0     1    75    0    2  0  0 100  0  0
 0  0      0 26029860   8356  76512    0    0     0     0    3   27  0  0 100  0  0
 0  0      0 26029860   8356  76512    0    0     0     0    7   39  0  0 100  0  0
```

1초간 3회 출력. 초당 문맥교환수
cat /proc/2166/status 로 자발적,
비자발적 문맥교환 비교가능

5.- Q&A

1. Nonpreemptive, preemptive scheduling 각각에 포함되는 스케줄링 기법에 대해 간단히 정리해주세요(다음 챕터 예습 겸)
2. 혹시 추추가 가능하면 scheduler는 잘 아는데 dispatcher는 생소해서 이게 어떤 건지 부가설명해주실 수 있나요

Preemptive scheduling - SRT, RR, Multilevel Queue

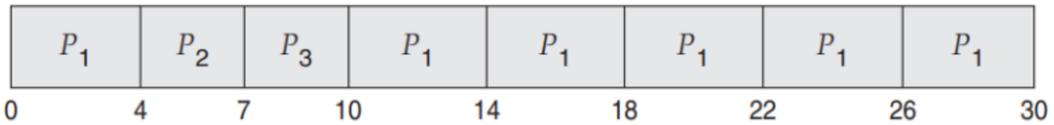
Nonpreemptive - FCFS, SJF, Priority

Process	Burst Time
P_1	24
P_2	3
P_3	3

FCFS 하고 비슷하지만 preemption이 추가되었다.

FCFS

Round Robin



Process	Burst Time	Priority
P_1	4	3
P_2	5	2
P_3	8	2
P_4	7	1
P_5	3	3

Priority