[Chapter 3]

소프트웨어융합학과/2020105742/한지훈

1. Program VS Process
   A. Program
      I. Secondary storage에 다운받아 설치
      II. 실행된 상태는 아니지만, 언제든지 실행할 수 있는 상태
   B. Process
      I. 프로그램이 실행된 상태
      II. Control flow(code segment 안에서 instruction에 대한 흐름)의 encapsulation
      III. Dynamic: 실행 중 input에 따라 real time으로 결과값 생성
      IV. basic unit: 실행, 스케줄링 관리의 가장 기본적인 단위
      V. process ID: 정지, 실행 중 등 상태에 따라 고유 ID 부여(스케줄링 위함)

2. Process Address Space
   A. Code segment
      I. Instruction 덩어리인 Code segment 저장
   B. static data
      I. 연산에 필요한 상숫값 저장
   C. stack
      I. 정적인 data 저장
      II. 명칭: Function call이 이루어지는 형태가 스택이어서 이름이 스택
      III. 문제: Callee가 동적할당 하려고 하면 불안정해짐 > 동적 data는 따로 관리 필요(Heap)
   D. heap
      I. 동적인 data 저장
   E. SP, BP
      I. SP(Stack pointer): Stack을 얼마나 사용하는지 표시
         - 초기값: main 함수만큼 할당
         - Function call시 SP가 그만큼 아래로 이동, BP(Base pointer)도 따라 이동
         - > 지역변수가 생기는 이유

3. State diagram
   A. New
      I. 개요: loader가 memory에 load를 하면 여러 사전작업을 하는 상태
      II. 변이(B): 사전작업 완료 후 Ready(B) 상태로 변이
   B. Ready
      I. 개요: 여러 사전작업이 완료되어 실행 준비가 완료된 상태
      II. 변이(C): 스케줄러는 Ready 상태 프로세스 여러 개 중 1개를 당장 CPU에 올라갈 수 있게 권한을 부여해줌 (Scheduler dispatch) > 선택된 프로세스는 Running(C) 상태로 변이
   C. Running
      I. 개요: 실행이 되고 있는 상태
      II. 변이(B): 아래 2가지 경우에 Ready(B) 상태로 변이
         - Interrupt 발생
         - Timer(CPU 사용 한계 초과시)
      III. 변이(D): Trap(어플의 I/O 요청)시 기다림 필요(synchronous) > Waiting(D) 상태로 변이

IV. 변이(E): 실행 완료, 오류 발생 등으로 인해 종료가 필요하면 Terminated(E) 상태로 변이

D. Waiting

I. 개요: Synchronous 시스템의 특성상 대기가 필요한 상태

II. 변이(B): 요청 서비스, I/O 등 끝나면 나중에 다시 실행되기 위해 Ready(B) 상태로 변이

E. Terminated

I. 개요: 실행 완료, 오류 발생 등으로 인해 종료되어 사라짐

4. PCB(Process Control Block)

A. 개요: 프로세스가 CPU에서 내려왔다가 다시 올라갈 때 마지막 상태 백업 해 놓은 구조체

I. 백업 위치: CPU의 레지스터

II. 작동: 백업 후 Dispatch시 레지스터에 restore하여 작업 재개

B. 포함 정보: Process state, Program counter, CPU scheduling information, ...

C. 생성 시기: Loader가 프로세스를 메모리에 load할 때

5. Context Switch

A. 개요: OS 관점에서 CPU의 사용 주체 계속 달라짐(각 프로세스의 control flow가 바뀜)

I. 발생 대상: Ready 상태인 2개 이상의 서로 다른 프로세스

II. PCB: 교차 시 기존 프로세스 PCB에 save, 새 프로세스 PCB에서 reload 진행

III. 과정: 커널 개입 > 기존 PCB save > 커널모드 수행 > 다른 PCB reload > 반복

B. 문제: 오버헤드 발생(Context switch는 필수가 아니라 Multiprogramming 구현을 위한 선택)

I. 특징: 없애는 건 불가능, 최소화 필요

II. 해결 방법: SW는 효과 적음, HW 활용(UltraSPARC - Multiple register sets)

6. Schedulers

A. Short-term scheduler(CPU scheduler): 기존 개념(Ready 상태 프로세스 중 누구를 CPU에 올릴 것인지)

B. Long-term scheduler: 여러 프로세스 중 누구를 메모리에 올려 ready 상태로 만들 것인지

C. Medium-term scheduler: 메모리에서 동작 중 공간 없을 시 하나를 스토리지로 내릴 때 누구를 내릴 것인지

7. Queueing diagram

A. 개요: 실행 중이 아닌 나머지 Ready/Waiting 상태의 프로세스도 Queue 통한 관리 필요

B. Ready 상태

I. Ready queue에 프로세스의 PCB 넣어 놓음

C. Waiting 상태

I. I/O 대기, 서비스/이벤트 대기 등 Case가 다양 > 세분화해서 관리

- Ex) mag tape unit, disk, terminal unit, ...

8. Operations on Processes

A. Process creation

I. fork()

- 과정:
  - 커널에서 새 PCB 생성
  - 새 address space 생성
  - Fork 수행한 프로세스의 PID 제외 나머지 부분 복사해서 새 프로세스 생성
  - 구분을 위해 다른 PID 부여

- PCB를 ready queue에 삽입

- 복사:
  - Child는 Parent의 PID 제외 모든 것 복사
  - 실행 상태도 복사(fork 생성, 아직 return하지 않은 상태)
  - Shallow copy > fork 중 A 파일 오픈하면 child fd도 A 가리킴(I/O, 파일 parent와 공유 가능)
- Return:
  - Parent: child의 PID return / Child: 0 return / Exception: -1
    > If/else로 return 값에 따라 같은 프로그램이 병렬적으로 동작 가능(멀티 프로세스)
- 활용:
  - 여러명의 클라이언트 요청 동시 처리 가능(Child: 요청 처리, Parent: 다른 요청 대기)
  - Web Server

B. Process execution

I. exec()

- 과정:
  - 현재 프로세스 정지
  - 프로그램 'prog'를 프로세스 code segment에 덮어씀
  - 시작을 위한 하드웨어, arg 초기화
  - PCB를 ready queue에 삽입
- 특징: 새 프로세스 생성하지는 않음
  > 프로세스 생성시: fork() + exec()

C. Process termination

I. exit(): Normal, 프로그램 정상종료를 커널에 요청(싹 다 반환)

II. _exit(): Normal, 그냥 종료(시스템 라이브러리, 커널 개발 말고 거의 안 씀)

III. abort(): Abnormal, 커널이 뭐 더 해줘야 할 게 있을 때

IV. wait():

- 역할: 자식 프로세스 종료될 때까지 기다림
  > 종료되면 정보 반환 시 PCB 일부 주요 정보 담고 있다가 부모 wait()에 정보 정리
  > 프로세스 간의 시간적 동기화에도 사용(자식 종료 후 부모 실행)
- 파생 상태
  - Zombie: parent가 wait하지 않았을 때 발생
    특징: 죽었지만 죽지 않은 상태
    의도적으로 사용하는 경우도 있어 꽤 볼 수 있음
  - Orphan: parent가 wait하지 않고, child보다 먼저 종료 시 발생
    특징: zombie의 특수한 경우
    Parent(연결고리)가 없음(문제)
    Handling: 커널 놀 때 트리 root를 부모로 지정하여 주기적으로 wait 걸어 지워줌

D. Cooperating processes

I. Inter-Process Communication (IPC): 프로세스 간에 data 주고받아야 할 때 사용

9. Windows Process Creation/Execution

A. CreateProcess()

I. 특징: fork(), exec() 통합 > 윈도우 프로세스는 부모/자식 관계가 없음(독립적)

10. Process Termination

A. Normal

I. main() return: main 함수가 모두 실행되고 난 후에 정상적으로 종료되는 것

II. exit(): 프로그램 정상 종료를 커널에 요청(파일, 메모리 등 싹 다 반환한 후에)

III. _exit(): 각종 반환 없이 그냥 종료(시스템 라이브러리, 커널 개발 제외하고는 쓸 일 거의 없음)

B. Abnormal

I. abort(): exit()의 기능 + 에러사항을 어떻게 핸들링 할 지까지 지정

II. signal: 강제 종료하는 경우(ex) 윈도우 작업관리자 > 강제 종료)

C. Zombie, Orphan

I. Zombie: 부모는 생존, 자식이 죽었지만 wait하지 않아 PID가 지워지지 않은 경우

- 특징: 의도적으로 사용하는 경우도 있음
- 해결법: 부모를 종료하면 자식 관련 데이터 삭제됨

II. Orphan: 자식은 생존, 부모가 wait하지 않고 죽어 부모-자식의 연결고리가 사라지는 경우

- 특징: zombie의 특수한 경우
- 해결법: 커널이 놀 때 트리 root를 부모로 지정하고 주기적으로 wait 걸어서 지워줌

11. C 프로그램 실행 과정

A. Kernel: exec()을 수행하여 C start-up routine을 호출

B. C start-up routine: 시스템 소프트웨어로, 실행을 위한 각종 밑작업을 수행한 후, main()을 호출

C. Main function: 코드가 실행되다가 다음의 상황 중 한가지가 발생

I. 사용자에 의해 _exit로 비정상 종료

II. User function을 call하여 수행, 완료되면 다시 main으로 return

III. Exit로 정상적인 종료가 되어 exit function을 호출

D. exit function: exit handler, I/O cleanup 등을 통해 뒷정리를 한 후, _exit()로 정상종료

12. Multi process

A. Google Chrome

I. Browser process: 렌더링에 필요한 파일, 사진 등 정보를 받기 위해 I/O 요청 해주는 프로세스

II. Renderer process: 웹페이지의 텍스트, 사진 등을 렌더링 해주는 프로세스

III. Plug-in process: 각종 plug-in 사용을 위한 프로세스

IV. 보안: Sandbox(브라우저와 동일, 시스템과 완전히 분리되어 test 해보고 이상하면 사전 차단 가능

B. mobile system

I. 과거: 2개의 프로세스만 존재

- foreground process: User Interface 등 직접 보여지는 프로세스
- background process: Memory, 실행 상태 등 뒤에서 돌아가는 프로세스

II. Android: 리눅스 기반으로 초창기부터 멀티 프로세스 가능했음

III. 아이폰: 초창기에는 음악 빼고 batch system으로 멀티 프로세스 불가, but 속도는 더 빨랐음

13. Inter-Process Communication (IPC)

A. message passing

I. 개요: 통신 요청 시 커널이 메모리 영역에 message queue 생성, A가 data 넘기면 커널이 신호를 B로 보냄

II. 장점: 개발자 입장에서 커널이 해주는 부분이 많기 때문에 code size가 상대적으로 작음

III. 단점: 커널이 계속 개입하여 커널 모드 진입횟수가 증가하여 오버헤드가 큼

B. shared memory

I. 개요: 통신 요청 시 커널은 비어 있는 메모리 일부를 shared memory로 할당해주고, 이후에는 개입하지 않음

II. 장점: 커널 모드 진입 횟수가 적어서 오버헤드가 작음

III. 단점: 커널 역할을 프로세스가 해줘야 하기 때문에 code size가 상대적으로 큼

IV. 주의사항: 완료 후 shared memory를 A, B에서 안지우면 garbage memory가 됨(사용 불가)

V. 동기화 문제: shared memory는 관리되지 않는 공유자원, control switch는 instruction 단위로 일어나기 때문에 문제가 생길 수 있음(Producer, Consumer의 in-out-in-out-... 순서가 깨지게 됨)

C. Sockets

I. 개요: 원격의 다른 PC와 통신을 할 수 있도록 해주는 API

II. 특징: 컴퓨터 프로그램과 다른 원격 컴퓨터의 프로세스 간에 통신을 위해 IPC가 필요함

III. 활용: 로컬 환경에서도 편의를 위해 프로세스 간에 socket 통해 연결하는 경우 있음

D. Remote Procedure Call (RPC)

I. 개요: 클라이언트에서 함수 호출 시 서버가 더 빠르면 원격으로 호출함수 이름, 변수 넘겨 원격 함수 실행