[Chapter 4]

소프트웨어융합학과/2020105742/한지훈

1. Process의 단점
   A. 메모리, OS 자원 등을 모두 포함하여 너무 무거움
   B. 새 프로세스 만드는 데 자원이 많이 들고, mode change가 자주 발생하여 오버헤드가 너무 큼

2. Thread
   A. 개요: 하나의 프로세스 안에서 control flow 상태 값은 병렬적으로 관리하고, 나머지 정보는 공유
   B. 장점: 멀티 스레드(자원 사용량이 적고, context switch 횟수가 줄어 효율적)
   C. 구성
      I. Control flow와 관련 있는 요소: 병렬적 관리(registers, stack)
         - Stack: 스레드 별로 stack에 나눠서 쌓임, 오버 플로우 방지 위한 guard space 존재
      II. Control flow와 관련 없는 요소: 전부 공유(code, data, file)

3. Web server with Multithread
   A. 개요: 하나의 프로세스 안에서 여러 개 스레드가 기존 부모/자식 프로세스의 역할을 대신함

4. Multi-Threads code
   A. #include <pthread.h>: POSIX 표준 thread 의미
   B. pthread_create: 새 thread 생성
      I. t tid: thread의 관리를 위한 각종 data를 패키징 형태로 모아놓은 구조체 선언
      II. (void *) DoCmd: 함수 포인터로, DoCmd로 점프하여 task 수행을 하기 위함
      III. (void *) cmd: DoCmd 결과값을 포인터로 반환, input parameter 1개 > 2개 이상시 구조체로 묶어 보내기
   C. pthread_join: unix의 wait와 같은 개념, 원본 thread에서 실행되어 나머지 스레드 DoCmd 끝나면 아래 실행
   D. #if ~ #endif: compile time에 처리되는 것으로(compile 할지 말지), conditional compilation이라 부름

5. Two Programs on Arduino
   A. Two Programs
      I. LED: loop() 통해 0.5초마다 LED 전구 전원 on/off를 반복
      II. BUZZER: loop() 통해 0.3초마다 도, 레, 미, 파의 4가지 음을 반복
   B. 합쳐보기
      I. 단순 loop(): 구현할 수 없음
      II. vTaskStartScheduler(): 멀티스레드를 통해 task(아두이노에서 프로세스보다 작은 단위) 2개 동시실행 가능

6. Multicore Programming
   A. single-core: Concurrent execution
      I. 개요: 여러 개 프로세스 간에 계속 왔다갔다 하면서 한 번에 하나의 프로세스 수행
   B. multicore: Parallel execution
      I. 개요: 서로 의존성 없으면 Concurrent 대비 core 개수만큼 배로 성능 증가(의존성 있으면 성능 상승 감소)
      II. Data parallelism: task보다 data를 어떻게 쪼개서 쓸지가 더 중요
      III. Task parallelism: control flow 나누는 것 자체가 더 중요, data는 다 공유
         - 예시: GPGPU(GPU 병렬 연산), image processing
      IV. 라이브러리: Task parallelism은 Pthreads가 유일, 나머지(OpenMP, GPGPU) 등은 전부 Data parallelism 특화

7. User-level threads VS Kernel-level Threads

A. 역사: Unix 초창기: thread 개념 X, 프로세스 단위 멀티프로그래밍괴 time sharing 개념만 존재)

- 10년 후 LWP(light weight process)개념 등장, 사용 위해서 구현 필요
- 기존 monolithic kernel과 thread를 결합하려고 시도했으나 어려웠음
- 커널과 독립적인 라이브러리 형태로 제공(POSIX Pthreads, Mach C-thread, Solaris threads)
  - 의미: 커널은 프로세스 안 스레드 있는지 여부 알 수 없음
- Kernel과 독립적으로(user level)에서 동작하여 User-level thread라 불림, 문제가 있었음
- Thread를 커널로 내림(Kernel-level Thread)
- User-level thread에 어떤 문제가 있어서 Kernel-level Thread로 바뀐 건지 이해 필요

B. User-level threads

I. 장점: 스레드가 kernel이 아닌 user space에서 돌아가기 때문에 kernel 오버헤드가 작음

II. 단점: 만약 프로세스의 일부 스레드에서 할 일이 많아도 다른 스레드에서 I/O 요청을 하면 그냥 내려버림

- 결국 kernel은 스레드들의 존재를 모르기 때문에 비효율적인 선택을 하게 됨

C. Kernel-level Threads

I. 장점: 커널은 프로세스 안 스레드들의 존재를 알기 때문에 이를 고려하여 효율적 동작 가능

II. 단점: 프로세스와 스레드를 이중으로 스케줄링하여 kernel 오버헤드가 큼

8. Multithreading Models

A. Many-to-One: User-level thread이며, kernel에는 한 개의 스레드만 존재하고 user space에 여러 개 존재

B. One-to-One: Kernel-level Thread이며, fork() 없이 멀티스레드를 사용하는 윈도우 등에 사용

C. Many-to-Many: 커널의 부하를 줄이기 위한 것으로, user/kernel space 모두에 다수의 스레드 존재

D. Two-level: Many-to-One과 One-to-One을 결합한 하이브리드 형태로, 유닉스 등에 사용

9. Fork() VS Thread

A. 문제: thread 여러 개 있는 상태에서 fork 실행 시 결과값이 무엇일지 애매한 경우 발생

B. 해결법: 프로세스가 fork 더 이상 하지 않는다는 가정하에 multi-thread 구현하자(불문율)

10. 각종 OS에서의 Pthreads

A. Pthreads (POSIX threads)

I. pthread_create: 새 스레드 생성, fork()와 같은 개념

II. void pthread_exit: 수행 스레드 종료, exit()와 같은 개념

III. int pthread_join: 스레드를 기다려서 동기화를 해주는 역할, wait()와 동일한 개념

B. Windows: CreateThread, ExitThread을 통해 스레드를 생성하고 종료

C. Java: Virtual machine을 사용하며, 'runnable'라는 인터페이스를 통해 스레드 사용

11. Threads Design Space

	Thread	Process
MS/DOS	1	1
older UNIXes	1	many
Mach, NT, Chorus, Linux, …	many	many
Java	many	1

- Java: Virtual machine 사용, 특수한 경우