Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 및 회로 설계 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 장성환
redmk1025@gmail.com

System call 부분을 보려면 man -s2 open 과 같이 본다. open 에 플래그를 추가하는 과정에서 O_EXCL (파일이 있으면 무시) 등 내용과 반대되는 것이 있으면 open 의 값이 0 보다 작아진다. (오류발생 여부 확인) 0644 는 파일 생성에 관련한 권한과 관련이 있다.	

Pipe communication ps 명령어는 process 의 state 를 보는 명령어에 해당한다. Ps -ef | grep bash | grep -v grep | awk ‘{print $2}’ PID - 고유한 프로세스 넘버 (민증과 같다)	ps -ef 는 컴퓨터에서 돌아가는 현재 프로세서들을 모두 보여준다. grep bash 는 bash 를 찾아라 grep -v grep 은 현재 수행하는 grep 은 제외하라. (bash 찾는 수행 제외) print $2 는 PID 를 보겠다는 것이다.

Tail command tail 명령어는 파일 내용의 마지막부터 읽을 때 주로 사용한	Tail -c 20 mytar.c (글자수) tail -n 1 /var/log/messages (라인 수) 이런식으로 내용을 전부 볼 필요가 없을 때 유용하다.

mbr
master boot record

운영체제는 magic number 를 가지고 있다.
맨 마지막에 510 바이트 위치에 매직 넘버를 삽입하여
이 부트코드가 리눅스 인지 윈도우 인지 나타낸다.

X86 시스템은 아직도 이 mbr 을 사용한다. (인텔은 하위호환 지원)
arm 은 안쓴다.

멀티 태스킹

cpu 란 놈은 뭘 관리해야 할까 ? 프로세스
프로세스는 cpu 의 추상화다. (가상의 cpu 이다.)

프로그램 코드를 보았을 때, a.out 이라는 실행 파일이 나왔다.
이러한 실행파일은 int main {~~~} 으로 생성하였다.

프로세스는 메인으로 만들어진 만든 프로그램 or 들의 집합 덩어리.
무한 루프를 돈다고 하였을 때, 메모리에 계속 상주하면서 실행되는 상태

DRAM 상에 이러한 정보들이 차곡 차곡 쌓여있다.
Int main 이라는 c 언어를 다시 까보면 기계어 어셈블리어 들이 들어있다.
이러한 기계어가 제어하는 것들이 범용 레지스터를 제어한다.

이러한 범용 레지스터를 가지고 가상메모리에 스택을 형성하면서 작업을 하였다.
Ps – ef 했을 때, 나오는 것들 엄청나게 많은 PID 들이 나왔다.

이러한 PID 들이 프로세스 이다.

cpu 안에는 범용 레지스터가 들어있다.
기계어를 동작 시킬때도 cpu 의 범용 레지스터를 사용한다.

프로세스가 여기에 있는데 어떻게 전부 실행될 수 있을까 ?
이것에 대한 답이 바로 프로세스는 cpu 의 추상화이다. 라는 것이다.

동시 다발적으로 동작하고 있느냐 ? → 싱글코어
병렬처리는 뭐지 ? → 멀티코어

컴퓨터 구조론에 따르면,
cpu 는 오로지 한 순간에 한가지 연산만 수행한다.

Context switch 가 이것에 해답이 될 수 있다.
Context switch 를 기반으로 멀티 태스킹이 가능하다.

예를 들어서 cpu clock 이 2Ghz (= f) 라고 하자.
주기는 T = 1/f
1 clock 이라는 것을 뭘 의미할까 ? 명령어 하나를 수행하는데 걸리는 시간.
즉, T = 1/(2*10^9) = 5*10^-10 즉 10 억분의 5 초 마다 한번씩 수행한다.

아주 빠른 속도로 프로세서들이 제어권을 넘겨주면서 cpu 를 사용한다면,
우리가 느끼지 못하는 순간에 모든 작업이 완료된다.
(제어권을 넘겨주는 것을 context switch 로 보면된다.)

A 프로세스와 B 프로세스가 다른 기계어 명령어를 가질때,
같은 레지스터를 쓰므로 프로세서의 제어권이 넘어갈 때, 연산 결과가 꼬일 염려가 있을까 ?

Task _struct{ // process 마다 생김
thread _struct * //해당 프로세스의 eax 등등 을 저장
thread _union *

태스크가 각각 자기만의 레지스터의 주소값을 가질 수 있는 것이다.

그래서 프로세스가 cpu 의 추상화.

프로세서가 cpu 의 제어권을 얻기 위해서 context switch 를 도입
중요한 작업을 우선처리 하기 위하여 우선순위 기법을 도입.

RTOS 는 우선순위가 중복되지 않는다.

멀티 코어는
wait queue
run queue
를 가지고 rq 에는 실행되는 프로세스가 저장되어 있다.
만일 동작 시간이 길어지면 wq 로 들어가게 되고,
우선순위를 판단하여 다시 rq 로 올리는 작업을 한다.
우선순위 오류 때문에 wq 에 있던 프로세스가 rq 로 못 올라가는 현상을 방지하기 위해서 처리못한 기간이 길어지면 rq 로 올려 버리는 기능도 있다.
(작업 못한 것들의 비율을 따져서 올림)

이러한 전체 동작 전반이 멀티 태스킹이다.

PIC
programmable interrupt controller

키보드, 장치, RTC

인터럽트가 들어오는 순간 rq 의 프로세스를 바로 wq 로 보내고
인터럽트의 작업을 우선처리 하고 wq 간 프로세서를 이어서 처리한다.

블록킹과 논 블록킹 문제 (myfifo.c) read 라는 시스템 콜은 블록킹 함수이다. 블록킹은 입력을 할때까지 제어권을 넘겨주지 않겠다는 것. 블록킹이 좋냐? 논블록킹이 좋냐 ? 아주 빠르게 통신해야 할때는 논 블록킹이 좋다. 반드시 순차적으로 진행되어야 하는 것은 블록킹이 좋다.	

ls – al /dev none - 일반파일 d – directory (blue) p – pipe c – character device b – block device socket device 는 b 로 아동. b 와 c 의 차이점 ? block 은 어떤 특정 단위를 가지고 움직임. character 는 순서가 있는 것 4kbyte 는 실제 메모리의 최소단위 (페이지) 키보드, 모니터는 반드시 입력한 순서가 지켜져야 한다.	

None blocking (myfifo2.c) 추가된 것은 fcntl() 함수로써 논 블럭킹 설정을 해준 것이다.	