TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 - 이상훈
gcccompil3r@gmail.com

학생 – 이우석
colre99@naver.com

[4/10(화) – 34 일차]

[chaper 3] (지난주)

* 런큐(Run queue) & 스케줄링(Scheduling) *

스케줄링: 여러개의 태스크들 중에서 다음번 수행시킬 태스크를 선택하여 **CPU** 라는 자원을 할당하는 과정
(다중 프로그래밍을 가능하게 하는 운영체제의 동작기법. 자원배정을 적절히 함으로써 시스템 성능 개선 가능)

런큐: 일반적인 운영체제는 스케줄링 작업 수행을 위해, 수행가능한 상태의 태스크를 자료구조를 통해 관리.
리눅스 에서는 이 자료구조를 런큐(Run queue)라고 한다.

* 리눅스의 태스크는 실시간 태스크 & 일반 태스크 로 나뉜다. (각각을 위해 별도의 스케줄링 알고리즘이 구현되어 있음)

[리눅스가 제공하는 140 단계의 우선순위]
실시간 태스크 : 0 부터 99 단계 사용. (숫자가 낮을수록 우선순위 나타냄)
일반 태스크: 100 부터 139 단계 사용.
→ !! 실시간 태스크는 항상 일반 태스크 보다 우선하여 실행된다. !!

실시간 태스크 스케줄링 (FIFO, RR, & DEADLINE)

task_struct 구조체는 policy, prio, rt, priority 등의 필드가 존재.
Policy 필드는 이 태스크가 어떤 스케줄링 정책을 사용하는지를 나타낸다.

리눅스의 태스크는 총 6 개의 정책이 존재. (**실시간** 태스크를 위해 **3** 개 + **일반** 태스크를 위해 **3** 개)
*** 실시간 태스크**를 위해서는 3 개의 정책: SCHED_FIFO, SCHED_RR, SCHED_DEADLINE 정책이 사용된다
*** 일반 태스크를** 위해서도 3 개의 정책: SCHED_NORMAL, SCHED_IDLE, SCHED_BATCH 정책이 사용된다.

*** 부가설명:**

<실시간>

SCHED_FIFO: first in first out(선입선출). Priority 값을 1 부터 99 까지 가질수 있고, sched_yield()호출 하거나, runnable 한 상태가 되면 priority 에 해당하는 대기 리스트의 맨뒤, 자신보다 더 높은 priority 를 가진 스레드에 의해서만 선점되고, 선점한 스레드가 CPU 를 놓아주면 계속 동작하는 정책.

SCHED_RR: 우선, 라운드 로빈(round robin)은 동등한 우선순위를 가진 스레드에게 동등한 간격으로 시간을 주는것을 말하고, 이 스케줄링 정책은 우선순위에 관계없이 일정한 시간을 할당해서 하는 정책을 말함.

SCHED_DEADLINE: 마감시간이 있는 스케줄링 정책으로 우선순위를 먼저 배정해서 실행하고 후속작업은 나중에 최소 시간으로

실행하게 하여 시간내에 동작시키도록 하는 정책.

<일반>

SCHED_NORMAL: 스레드는 priority 값을 0 만 가질수 있고, time share 하면서 동등하게 스케줄링 하는 정책.

SCHED_IDLE: 중요하지 않은 일을 수행하는 태스크가 CPU 를 점유 하는것을 막기위해 가장낮은 우선순위로 스케줄링 되게한다.

SCHED_BATCH: 사용자와의 상호 작용이 없는 CPU 중심의 일괄 작업(batch job) 태스크를 위한 정책.

[Chapter 4] – 메모리 관리 (배운거 정리)

물리 메모리의 한계를 극복하기 위해 가상 메모리 사용.

가상 메모리는 물리 메모리의 크기와 관계없이 가상적인 주소 공간을 사용자 태스크에게 제공한다.
(페이징을 통해서 뒤에서 관리한다.)

SMP(Symmetric Multi processor)
:모든 CPU 가 메모리와 입출력 버스 등을 공유하는 구조

NUMA(Non-Uniform Memory Access)
:CPU 들을 몇개의 그룹으로 나누고 각각의 그룹에게 별도의 지역 메모를 주는 구조 (병목현상 해결위해 사용)

[2-1] Node

뱅크(bank)
: 리눅스에서 접근속도가 같은 메모리의 집합

노드(node)
: 리눅스에서 뱅크를 표현하는 구조

* 리눅스는 하드웨어 시스템에 관계없이 노드라는 일관된 자료구조를 통해서 전체 물리 메모리를 접근할 수 있게 되는 것. 복수 개의 노드는 리스트를 통해 관리된다. 이는 pgdat_list 라는 이름의 배열을 통해 접근가능 하다.

하나의 노드는 pg_data_t 구조체를 통해 표현된다. 이 구조체는 해당노드에 속해있는 물리 메모리의 실제양(node_present_pages)이나, 해당 물리메모리가 메모리 맵의 몇번지에 위치하고 있는지를 나타내기 위한 변수(node_start_pfn) 등이 정의되어 있다.

Cache 를 활용하면 높은 성능을 얻을 수 있다.
→ 가까운 곳의 위치한 메모리를 사용

[2-2] Zone

node 를 관리하는 zone.(노드에 존재하는 물리메모리 중에) (node 의 일부분을 따로 관리할 수 있도록 자료구조를 만듦)

ex) ZONE_DMA, ZONE_DMA32
(DMA: Direct Memory Access) 직접 메모리 접근.
:주변장치들이 메모리에 직접 접근하여 읽거나 쓸수 있도록 하는 기능으로, 컴퓨터 내부의 버스가 지원하는 기능.

-ZONE_HIGHMEM-
:커널의 가상주소공간 과 1:1 로 연결해주고, 나머지 부분은 필요할때 동적으로 연결하여 사용하는 구조.
(물리메모리가 1GB 이상이라면, 896MB 까지만)

여기에서 커널의 가상주소공간과 1:1 로 연결 에서 물리메모리에 접근하려면 먼저 페이징을 해줘야 한다.

그리고, 나머지 부분은 필요할때 동적으로 연결. 이부분은 간접참조. (주소를 참조해서 매칭)

[2-3] Page frame

3. Buddy 와 Slab

[3-1] 버디 할당자(Buddy Allocator)

nptl 핵심:
커널전용 프로그램 , 사용자의 프로그램 이 1 대 1. 프로그램 사이즈크면 캐시가 깨짐 = 성능저하, 메모리 올리기힘듬. 캐시활용 저하. 기존의 1 대 n 모델을 1 대 1 모델로 바꾸고, 프로그램 사이즈에 제한을 둔다. 커널스택이란걸 만들어서 유저랑 통신할수 있도록 커널스택에서 활용. 커널스택은 스레드_유니온 이 커널스택. 즉 nptl 의 성능이 빨라졌다.

Deadline 은 정적 / 사용자 스케줄링은 동적. 실제시간은 달라도, 가상시간은 같아야 한다.