TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 : Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 : 황수정
sue100012@naver.com
2 개월 차 시험 오답노트

[임베디드 애플리케이션 구현1] 파이프 통신을 구현하고 c type.c라고 입력할 경우 현재 위치의 디렉토리에 type.c 파일을 생성하도록 프로그래밍하시오.

```
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

int flag;

char *check_text(char *text) //c type.c라고 입력했는지 확인하는 함수
{
	int i;
	static char filename[1024];
	int text_len = strlen(text);
	if(text[0] != 'c' && text[1] != ' ') //'c '로 시작하는 것이 아닐 경우
		return NULL;
	if(text[text_len - 1] != 'c' && text[text_len - 2] != '.') //'.c'로 끝나지 않을 경우
		return NULL;
	for(i = 2; i < text_len - 2; i++) //중간 값 확인
	{
		if(text[i] == ' ' || text[i] == '₩t') //띄어쓰기가 있을 경우
			return NULL;
		filename[i - 2] = text[i]; // text를 filename에 넣는다. 2를 빼는 것은 '.c'를 붙이기 위함?
	}
	strcat(filename, ".c"); // strcat(복사 받을 변수, 복사할 변수) filename에 .c를 덧붙이는 것
	return filename;
}

int main(void)
{
	int fo;
	int fd, ret;
	char buf[1024];
	char *string = NULL;
	fd = open("myfifo", O_RDWR); //파이프 통신을 할 것을 열음.
	fcntl(0, F_SETFL, O_NONBLOCK); //fcntl 함수는 파일 특성을 조정하는 함수.
	fcntl(fd, F_SETFL, O_NONBLOCK);
	for(;;)
	{
		if((ret = read(0, buf, sizeof(buf))) > 0)
		{
			buf[ret - 1] = 0;
```

```
                printf("Keyboard Input : [%s]₩n", buf);
                string = check_text(buf);
                printf("String : %s₩n", string);
        }
        if((ret = read(fd, buf, sizeof(buf))) > 0)
        {
                buf[ret - 1] = 0;
                printf("Pipe Input : [%s]₩n", buf);
                string = check_text(buf);
                printf("String : %s₩n", string);
        }
        fo = open(string, O_WRONLY | O_CREAT | O_TRUNC, 0644);
        close(fo);
    }
    close(fd);
    return 0;
}
```

> 파이프 통신도 다른 네트워크 프로그래밍처럼 주고 받는 것이 필요하다고 생각해서 코드를 2개 만들었습니다. 선생님 코드를 보고 아직 개념도 제대로 이해 못하고 있다는 것을 깨달았습니다.

[임베디드 애플리케이션 구현 2] 369 게임을 작성하시오. 2초내에 값을 입력하게 하시오. 박수를 쳐야 하는 경우를 Ctrl + C를 누르도록 한다. 2초내에 값을 입력하지 못할 경우 게임이 오버되게 한다. Ctrl + C를 누르면 "Clap!"이라는 문자열이 출력되게 한다.

- 선생님 코드를 분석하려다가.. 조금 더 손 봐서 다시 작성해 보려고 적지 않았습니다.. 더 공부해서 작성해보겠습니다.

[임베디드 애플리케이션 구현 7] Page Fault 가 발생했을때 운영체제가 어떻게 동작하는지 기술하시오.

Page Fault 는 가상 메모리를 물리 메모리로 변환하는 도중 물리 메모리에 접근했더니 할당된 페이지가 없을 경우 발생한다. 이것이 발생하면 현재 수행중이던 ip(pc) 레지스터를 저장하고 페이지에 대한 쓰기 권한을 가지고 있다면 Page Fault Handler 를 구동시켜서 페이지를 할당하고 **저장해놨던 ip(pc) 를 복원하여 다시 Page Fault 가 발생했던 루틴을 구동**시킨다. 만약 쓰기 권한이 없다면 Segmentation Fault 를 발생시킨다.

- 동작 과정을 적지 않아, 오답노트를 진행하였습니다. 단순하게 동작을 현재 가상 메모리에 페이지가 존재하는지 확인 후, 없으면 프로세스의 중단으로 잘못 알고 있었다는 것을 알게 되었습니다.. 쓰기 권한이 유무가 운영체제 동작을 달리 한다는 것을 알게 되었습니다. CPU는 물리 메모리를 확인하여 페이지가 없으면 인터럽트를 발생시켜 운영체제에 알린 후, 페이지 테이블을 갱신해 다시 구동 시킨다는 것을 알았습니다.

[임베디드 애플리케이션 구현11] 리눅스 커널 소스에 보면 current라는 것이 보인다. 이것이 무엇을 의미하

는 것인지 적으시오. 커널 소스 코드와 함께 기술하시오.
먼저 vi -t current 로 검색하면 아래 헤더 파일에 x86 에 한하여 관련 정보를 확인할 수 있다.
arch/x86/include/asm/current.h
여기서 get_current() 매크로를 살펴보면 ARM 의 경우에는 아래 파일에
include/asm-generic/current.h
thread_info->task 를 확인할 수 있다. x86 의 경우에는 동일한 파일 위치에서 this_cpu_read_stable() 함수에 의해 동작한다.
이 부분을 살펴보면 아래 파일
arch/x86/include/asm/percpu.h 에서
percpu_stable_op("mov", var) 매크로를 통해 관리됨을 볼 수 있다. Intel 방식의 특유의 세그먼트 레지스터를 사용하여 관리하는 것을 볼 수 있는 부분이다.

[작성 답안] current라는 매크로는 커널 내부에 정의되어 있는 매크로로써 현재 태스크의 task_struct 구조체를 가리 킬 수 있게 해주며(현재 구동중인 task의 task_struct의 포인터), task_tgid_vnr()은 해당 task_struct 구조체의 tgid 필드를 리턴한다. 따라서 이 함수는 task_struct 구조체의 tgid 필드 값을 리턴하는 단순한 함수라고 볼 수 있다.

- Current에 대해서만 적고 정확히 어디에 위치하고 있는지 기입하지 않아 오답노트를 진행했습니다. Current를 보고 나오는 함수를 기입하고 리턴하는 값만 기입하였습니다.

[임베디드 애플리케이션 구현 14] 리눅스 커널의 스케쥴링 정책중 Deadline 방식에 대해 기술하시오.
리눅스 커널의 Deadline Scheduler 는 Dario Faggioli 에 의해 2013 년 3 월 Linux Kernel v3.14 에서 소개되었다.
Deadline Scheduler 는 **EDF + CBS 알고리즘 기반으로 동작**한다. 임베디드 개발자들이 주로 원하는 기능이지만 토발즈가 별로 탐탁치 않아 한다. (**CPU Scheduler 만으로 Real-Time 을 해결할 수 없다고 생각**해서 그러함)
Task 마다 주어진 주기를 가지고 실행되는 것을 보장해야 한다. 이 Task 의 우선 순위는 종종 변동되는데, 만료 시각이 먼저 다가 오는 Task 에 우선 처리할 기회가 주어진다. 한 마디로 지금 당장 급한것부터 처리하자는 의미다.

- 어떠한 알고리즘을 기반으로 작동하는지와 왜 생성되었는지를 기입하지 않아 오답노트를 진행하였습니다.

[임베디드 애플리케이션 구현 17] 현재 4개의 CPU(0, 1, 2, 3)가 있고 각각의 RQ에는 1, 2개의 프로세스가 위치한다. 이 경우 2번 CPU에 있는 부모가 fork()를 수행하여 Task를 만들어냈다. 이 Task는 어디 에 위치하는 것이 좋을까 ? 그리고 그 이유를 적으시오.

2번에 위치하는 것이 좋다. fork()는 같은 Task를 복사하게 된다. 그리고 다시 그 코드를 사용할 것이라면 CPU의 **Instruction Cache, Data Cache를 재활용하는 것이 최고**다. 결론적으로 **Cache Affinity를 적극 활용**하자는 것이다.

- 캐시를 친화력에 의해 어디에 위치하는 것이 좋은지에 대한 이론은 기입하였으나, CPU의 위치는 기입하지 않아 오답노트를 진행했습니다.

[임베디드 애플리케이션 구현 18] 위 문제에서 이번에는 0, 1, 3에 매우 많은 프로세스가 존재한다. 이 경우 3번에서 fork()를 수행하여 Task를 만들었다. 이 Task는 어디에 위치하는 것이 좋을까 ? 역시 이유를 적으시오.

2번에 위치하는 것이 좋다. 이번에는 **주어진 시간내에 0, 1, 3 은 Scheduling 을 수행할 수 없기 때문에** 오히려 상대적으로 Task가 적은 2번에 배치하는 것이 좋다. Cache Affinity를 포기할지라도 아예 Task 를 동작시킬 기회가 없는것보다 좋기 때문이다.

- 부하균등에 대해서 적었으나 어떤 CPU의 위치인지는 기입하지 않아 오답노트를 진행하였습니다.

[임베디드 애플리케이션 구현 19] UMA와 NUMA에 대해 기술하고 Kernel에서 이들을 어떠한 방식으로 관리하는지 기술하시오. 커널 내부의 소스 코드와 함께 기술하도록 하시오.

메모리 접근 속도가 같은 것을 Bank라 한다. 그리고 **Kernel에서 이들은 Node**라 한다. UMA는 모든 CPU가 메모리 접근 속도가 같은 SMP와 같은것을 의미한다. NUMA는 CPU 마다 메모리 접근 속도가 다른 Intel의 i 계열의 CPU군을 의미한다. Linux Kernel에선 이를 **contig_page_data 를 통해 전역변수로 관리**한다. 그리고 **UMA의 경우엔 Node가 1개인데 pglist_data 구조체로 표현**된다. NUMA의 경우엔 pglist_data가 여러개 연결되어 있다.

- UMA와 NUMA에 대해서는 기입하였으나 내부 소스코드와 같이 기입하지 않았기에 오답노트를 진행하였습니다.

[임베디드 애플리케이션 구현 20] Kernel의 Scheduling Mechanism에서 Static Priority와 Dynamic Priority 번호가 어떻게 되는지 적으시오.
Static Priority : 1 ~ 99
Dynamic Priority : 100 ~ 139

- 이론을 기입하면서 정확히 정리하여서 답을 작성하지는 않아 오답노트를 작성하였습니다.

[임베디드 애플리케이션 구현 21] ZONE_HIGHMEM 에 대해 아는대로 기술하시오.

[임베디드 애플링케이션 구현 22] 물리 메모리의 최소 단위를 무엇이라고 하며 크기가 얼마인가 ? 그리고 이러한 개념을 SW 적으로 구현한 구조체의 이름은 무엇인가 ?
물리 메모리의 최소 단위는 Page Frame이라 하며 이것을 **SW 적 개념으로 구현한 구조체의 이름은 page**다.

- SW적 개념으로 구현한 구조체의 이름은 기입하지 않아서 오답노트를 하였습니다.

[임베디드 애플리케이션 구현 23] Linux Kernel은 외부 단편화와 메모리 부하를 최소화하기 위해 Buddy 할당자를 사용한다. Buddy 할당자의 Algorithm을 자세히 기술하시오.
[임베디드 애플리케이션 구현24] 위 문제에 이어 내부 단편화를 최소화 하기 위해 Buddy에서 Page를 받아 Slab 할당자를 사용한다. Slab 할당자는 어떤식으로 관리되는지 기술하시오.

- Buddy 와 Slab 할당자는 좀 더 공부 후에 정리하려고 비워두었습니다..

[임베디드 애플리케이션 구현 25] Kernel은 Memory Management를 수행하기 위해 VM(가상 메모리)를 관리한다. 가상 메모리의 구조인 Stack, Heap, Data, Text는 어디에 기록되는가 ? (Task 구조체의 어떠한 구조체가 이를 표현하는지 기술하시오)
task_struct 구조체에 있는 mm_struct 내에 start_code, end_code등으로 기록됨.

- mm_struct 까지만 기입하고 start_code, end_code등으로 기록되는 것은 작성하지 않아 오답노트를 진행하였습니다.

[임베디드 애플리케이션 구현26] 위의 문제에서 Stack, Heap, Data, Text등 서로 같은 속성을 가진 Page를 모아서 관리하기 위한 구조체 무엇인가 ? (역시 Task 구조체의 어떠한 구조체에서 어떠한 식으로 연결되는지 기술하시오)
task_struct 내에 **mm_struct 포인터를 따라가 보면 vm_area_struct 구조체가** 있다. 이 녀석이 서로 같은 Segment 들을 모아서 관리한다.

- mm_struct 밑의 변수를 찾아서 그것만 기입하고 어떤 식으로 연결되는지는 기입하지 않아 오답노트를 진행하였습니다.