TI DSP, MCU 및 Xilinx Zynq FPGA
프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 하성용
accept0108@naver.com

33 일차
p53
fork()와 vfork()의 차이점

exec:
일반적으로 실행 혹은 구동시키기위해서 사용
단점 : 기존메모리 날아감

exec 을 하게되면 뒤에 것을 실행 못하게됨
그래서 fork 를 하게되고 자식을 exec

그걸로 확인했던 예제가 시스템에 데이트 집어넣는거
mysystem 에 shell -date 실행

그런데 fork 를하고 exec 을하면 문제가 생기는데
문제는 fork 하게되면 메모리를 부모의 메모리를 복사함
부모의 메모리를 복사하는데 거기서 exec 을하게되면 메모리를 덮어쓰게됨
그렇게된다는건 쓸데없는 복사를 하게된다는것

어떻게 넣어야할까 프로세스정보, 구상한것 같이 꼭 필요한정보만
넣으면 어떨까해서 나온게 vfork

exec 만 사용할때 쓰는게 vfork 로
사용법은 fork 와 같음

p54
만약 쓰레드의 생성이었다면 수행결과에 어떤 차이가 있을까?

예제소스
```
#define _GNU_SOURCE //맨꼭대기 있어야 구동되는 헤더파일
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include<sched.h>

int g=2;

int sub_func(void *arg)
{
        g++;
        printf("PID(%d):Child g=%d \n",getpid(), g);
        sleep(2);
        return 0;
}

int main(void)
{
        int pid;
        int child_stack[4096];
        int l=3;
        printf("PID(%d) : Parent g=%d, l=%d \n",getpid(), g, l);
        clone (sub_func, (void *)(child_stack+4095),
                        CLONE_VM|CLONE_THREAD|CLONE_SIGHAND, NULL);
        sleep(1);
        printf("PID(%d) : Parent g=%d, l=%d \n",getpid(),g,l);
        return 0;
}
```
```
PID(4327) : Parent g=2, l=3
PID(4327):Child g=3
PID(4327) : Parent g=3, l=3
```

clone 이란?
#define _GNU_SOURCE 헤더없이는 구동이안되게되있음

clone 의 성질은
쓰레드도 될수있고 프로세스가 될수도있음
옵션을 어떻게주냐에 따라 다름

맨처음 뿌렸을때 전역변수 2 지역변수 3
자식을뿌리면서 전역변수 3 패런트가 3
fork 로 찍어보면 2 로 나옴

쓰레드는 메모리공유해서 자식의 변수와 부모의 변수가 같음

void+4095 // 쓰레드도 sub_func 이라는걸 실행하기위해 필요함
*arg //에는 아무값도 들어가지 않음
쓰레드를 구동하기위해 필요한것
왜 4096 이 아닌 4095 인지= 배열이 0 부터 시작이기때문에
왜 4096 이냐면 물리메모리 최소단위가 4k(4096)이기때문에
한번에 4096 받는게 성능상으로 좋음

클론_VM //가상메모리 쓸거라는것
thread//쓰레드로 만들겟다는것
시그널핸드//시그널하겠다는것
NULL //옵션

vi -해서 커런트들어가보기

18 번째
vi -t current
18

```
 1 #ifndef __ASM_AVR32_CURRENT_H
 2 #define __ASM_AVR32_CURRENT_H
 3
 4 #include <linux/thread_info.h>
 5
 6 struct task_struct;
 7
 8 inline static struct task_struct * get_current(void)
 9 {
10     return current_thread_info()→task; //
11 }
12
13 #define current get_current()
14
15 #endif /* __ASM_AVR32_CURRENT_H */
~
```

스위치 2 라는걸 가정
스위치 이후에 나오는 내용들을 따라간다고
즉 컨텍트스위칭을 쓴다는거

cpu domain 이 어디있는지 알아야 불필요한 캐시를 최대한 줄일수있음

struct cpu_context 스위칭을 할때 필요한 정보

arm 에서 사용하는

grep -rn “getpid” ./ grep SYSCALL

```
yong@yong-Z20NH-AS51B5U:~/kernel/linux-4.4$ grep -rn "getpid" |grep SYSCALL
tags:1284132:SYSCALL_DEFINE0    kernel/sys.c    /^SYSCALL_DEFINE0(getpid)$/;"  f
include/uapi/asm-generic/unistd.h:488:__SYSCALL(__NR_getpid, sys_getpid)
arch/arm/include/uapi/asm/unistd.h:48:#define __NR_getpid                     (
__NR_SYSCALL_BASE+ 20)
arch/um/os-Linux/start_up.c:183:        if (PT_SYSCALL_NR(regs) != __NR_getpid)
{
arch/um/os-Linux/start_up.c:185:                        "expected %d...", PT_S
YSCALL_NR(regs), __NR_getpid);
arch/um/os-Linux/start_up.c:189:        n = ptrace(PTRACE_POKEUSER, pid, PT_SYSC
ALL_RET_OFFSET, os_getpid());
arch/powerpc/include/asm/systbl.h:26:SYSCALL_SPU(getpid)
arch/s390/kernel/syscalls.S:31:SYSCALL(sys_getpid,sys_getpid)                   /
* 20 */
arch/xtensa/include/uapi/asm/unistd.h:271:__SYSCALL(120, sys_getpid, 0)
arch/arm64/include/asm/unistd32.h:65:__SYSCALL(__NR_getpid, sys_getpid)
kernel/sys.c:830:SYSCALL_DEFINE0(getpid)
```

task_tgid_vnrl(current)

/getpid

겟피아이디는 결론적으로 pid 값을 얻어온다는거

ns 를 통해서 얻어올수있는게 pid 숫자번호

getpid 도 퍼런트를 기반으로 동작

좀 단순화하기위해선 퍼런트가 가리키는 pid

return 퍼런트->pid

p61.c

```
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include<linux/unistd.h>

int main(void)
{
        int pid;

        printf("before fork \n \n");

        if((pid == fork()) < 0 ) {
                printf("fork error \n");
                exit(-2);
        }else if (pid ==0) {
                printf("TGID(%d), PID(%d):Child
\n",getpid(),syscall(__NR_gettid));
        }else {
                printf("TGID(%d), PID(%d):Parent \n",
getpid(),syscall(__NR_gettid));
                sleep(2);
        }
        printf("after fork \n \n");

        return 0;
}
```

```
before fork

TGID(5734), PID(5734):Child
after fork

TGID(5735), PID(5735):Child
after fork
```

fork와 pid, tgid

pgd란? 페이지 글로벌 디렉토리 약자
페이지 디렉토리 테이블
페이지 디렉토리 시작주소

p62.c

```
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include<linux/unistd.h>

int main(void)
{
        int pid;

        printf("before vfork \n \n");

        if((pid = vfork()) < 0) {
                printf("fork error \n");
                exit(-2);
        }else if(pid==0){
                printf("TGID(%d),PID(%d):Child \n",getpid(),
syscall(__NR_gettid));
                _exit(0);
        }else {
                printf("TGID(%d), PID(%d):Parent \n",getpid(),
syscall(__NR_gettid));
        }
        printf("after vfork \n \n");

        return 0;
}
```

```
before vfork

TGID(5821),PID(5821):Child
TGID(5820), PID(5820):Parent
after vfork
```

vfork와 pid, tgid

thread_union
thread info 안에는 뭐가 들어있었나
현재 커널에서 어떤 프로세스가 구동된다곳했을때 그 프로세스 커널자신의 정보를
current 가 가리키는 task_struct 로 얻을수있었다?
thread_union = 커널 스택
이 context 들을 다 어디서 관리? thread_union

하드웨어 컨텍스트란?
레지스터정보

레지스터정보가 중요한 이유?
인텔 기계어생각했을때 프로세스 a,b 돌아갔을때
레지스터를 둘이 같이 사용하면 크리티컬섹션이 꼬여서
나온게 컨텍스트스위칭

p.68
int on_rq 있으면 1 없으면 0 =>task_struct

prio : 스케줄링 알고리즘
si : 일반적인스케줄링
rt : 리얼타임 스케줄링
sched

signal_struct

signal
sighand : 액션같은걸 관리

task_struct 에 대부분의 커널기능들이 들어있음

sigpanding // 어떤걸 지연시키고싶을때쓰는거
memory information //가상메모리 세그먼트, pgd(페이지 디렉토리의 시작주소, 10 비트 10 비트 12 비트 쪼개서 수행, 레드블랙 수행,vm_struct->가상메모리관리하는)

file information 이란? files_struct

→fork() 하면 어디로 가는지?
[CPU]AAA (A) ← 여기로감 (이유: 다른 cpu 로 가게되면 기존정보가 없어서 캐시를 더 소모하기때문
[CPU]BB
[CPU]C

다음 fork 는 어디로→C
이유: 캐시가 너무많아서 Load Balancing,부하분산으로 가장 적은 캐시를 가진곳으로 보냄
[CPU]AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[CPU]BB
[CPU]C (A)

Heterogeneous
Architecture
이기종아키텍처 (최신 cpu)

fault info: 폴트제어를 위해 존재함 . 트랩이 무엇이 들어왔는지 알아야 동작도 할수있음
페이지폴트가 발생하면 메모리가 없으니까 페이지를 할당해줌
페이지 핸들러가 동작해서 유저에서 구동됬는지 커널에서 구동됫는지 확인하는데
커널에서 구동된거면 페이지를 할당해줌
유저에서 접근하면 권한이 없다고 세그멘테이션폴트 나옴

sys_fork 분석 못했습니다

NPTL → 개념
NPTL 이란?
초창기 리눅스는 포직스 쓰레드 api 를 유저 레벨에서 구현했다.포직스 쓰레드 api 를 부르면 리눅스 쓰레드 씨스템 콜이 호출되는 형식으로(clone()이나 fork()), 그런데 그렇게 하니 쓰레드 성능이 안나오고, 표준과 안 맞아 떨어지는 문제가 있어서 포직스 쓰레드 api 를 커널 레벨에서 구현하기 시작했었는데, 그 당시에 구현체가 2 개(NGPT, NPTL)있었는데 그중 NPTL 이 살아남아서, 리눅스 포직스 쓰레드 api 공식 구현은 NPTL 이 되었다