A2---

Start bunch of child processes using fork()

Token Ring Network System -Strategy

Compression -> let’s say there’s a multiple words ‘street’ in the file.

Then it replaces ‘street’ to \*(example) for the file so the size can be smaller.

Signals ->

1 HUP hangup: program is terminated gracefully. To terminal

2 INT interrupt

3 QUIT

4 ILL illegal instruction (wrong binary file)

5 TRAP trace trap ->

9 KILL kill process (different than quit, it doesn’t get interrupted. It will kill it nomatter what)

7 BUS bus error

11 SEGV segmentation violation

30 PWR power failure

15 TERM terminate process

10 USR1 user defined #1

12 USR2 user defined #2

Signal Handler’s body could be anything you like(you can code)

Signal returns the old signalHandler

So singal(SIGINT, signalHandler); -> settings for different signals.

Function name works like a pointer ( address)

&Function , Function -> both work.

Scheduling - > figuring out what process to be run next.

Time-slice used up :

while(1) - > it’s not gonna let users input

there’s time-slice limit ( maximum time of a process )

alarm-clock handler -> kicks in after the time-slice hits the limit

nice (negative) - > 빨리 해줘 (Priority high)

nice(positive) - > I am willing to wait ( priority low)

command ps -> shows all processes

batch - > is usually database

real-time - > It can predict how long it’s gonna take to finish a certain process. Make sure no one is waiting too long. Cuz everyone is equally important. strategy

How long did you spend on the processor last time - > priority queue 기준 !

어느쪽에다 넣을지 이걸 이용해 결정함.

-----------------------------------memory ------------------------------------------

BSS -> end of the static section / beginning of the dynamic data section

Fork() ->

Parent gets the value of its child

Child gets 0 from return

We don’t need to use critical section in fork. Because it doesn’t really share a lot of things. Most data is copied not shared!

Bu in thread, we need to use critical section because most data is shared

Only the stack is copied. Others are shared

Process image and logical pages -> physical 한곳에 아무대나 넣어도 logical 을 잘 짜면 막 섞여있어도 언제든지 찾아서 “논리적” 으로 연결되어있는 것 처럼 쓸수있다.

512(10) = 200(16)

Shift 1st 2nd 4th 8th

## ✅ ****1.** 페이지 크기**(Page Size)** 변환**

문제에서 주어진 페이지 크기는:

51210​bytes

이를 16진수로 변환하면:

51210​=0x20016​

### 📋 **이해하기**:****

* 51210​ → 29이므로 \*\*9비트의 오프셋(offset)\*\*을 사용한다는 의미입니다.

## ✅ ****2.** 가상 주소 분할 **(Virtual Address Breakdown)****

문제에서 주어진 가상 주소:

0x072E

이를 페이지 번호와 오프셋으로 나눕니다.

### 🔹 ****(1)** 페이지 번호 계산**

0x072E÷0x200=0x3

* 0x200=512이므로, 페이지 크기로 나눈 몫이 \*\*페이지 번호(Page Number)\*\*입니다.
* 결과적으로 **페이지 번호 = 0x3 (3)**

### 🔹 ****(2)** 오프셋 계산**

0x072Emod0x200=0x12E

* 나머지 값이 페이지 내부의 \*\*오프셋(Offset)\*\*입니다.
* **오프셋 = 0x12E (302)**

## ✅ ****3.** 페이지 테이블**(Page Table)** 조회**

가상 주소를 물리 주소로 변환하려면 \*\*페이지 테이블(Page Table)\*\*을 참조해야 합니다.

* **가상 페이지 번호 = 0x3**을 찾습니다.
* 페이지 테이블에서 **Page 0x3 → Frame 0xD**로 매핑되어 있습니다.

## ✅ ****4.** 물리 주소 계산 **(Physical Address Calculation)****

공식은 다음과 같습니다:

Physical Address=(Frame Number×Page Size)+Offset

### 🔹 **계산 단계**

1. **프레임 번호(Frame Number):** 0xD=13
2. **페이지 크기(Page Size):** 0x200=512

0xD×0x200=0x1A00

이제 오프셋을 더합니다:

0x1A00+0x12E=0x1B2E

## 🚀 **최종 답**:****

0x1B2E​