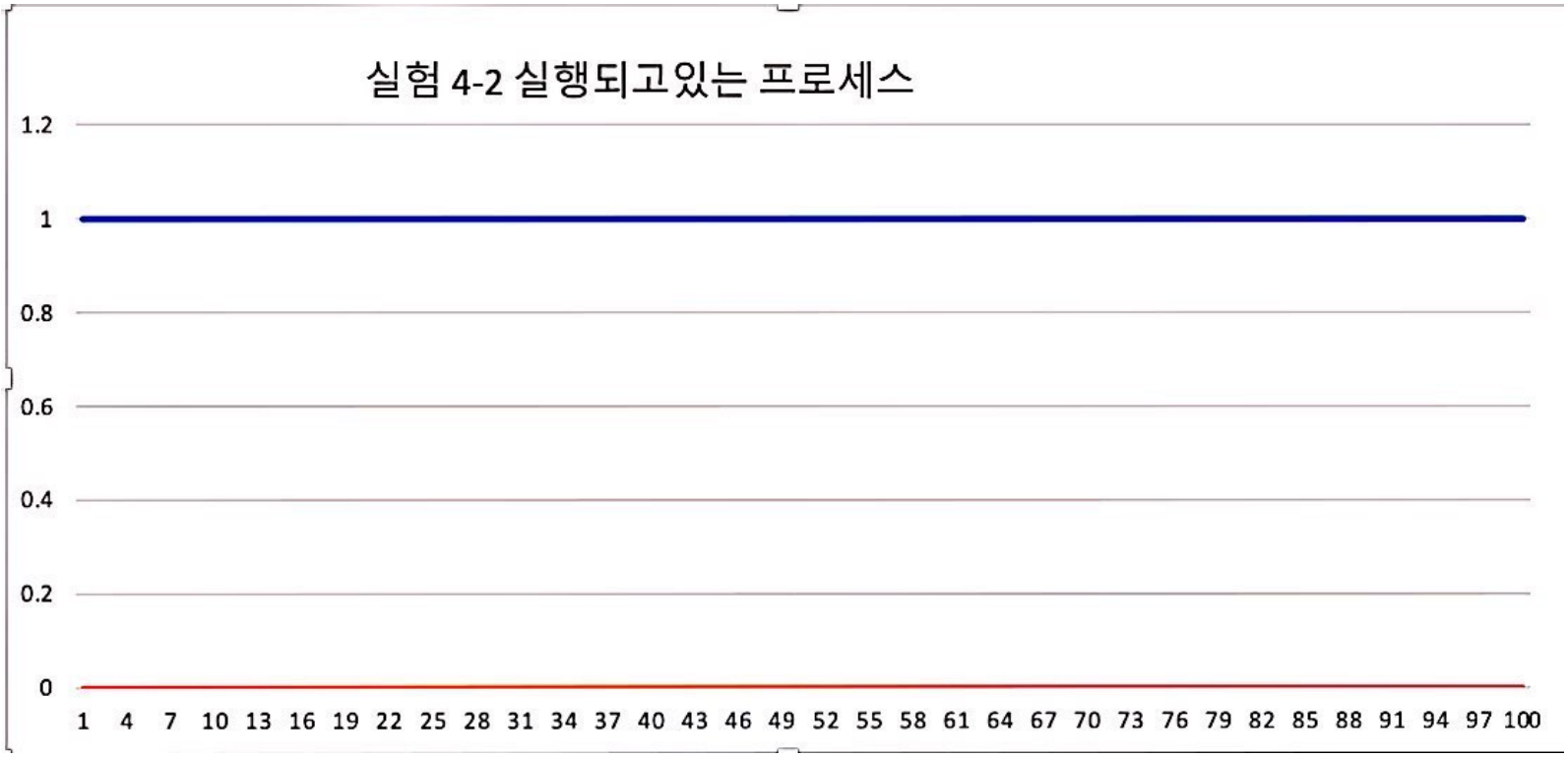
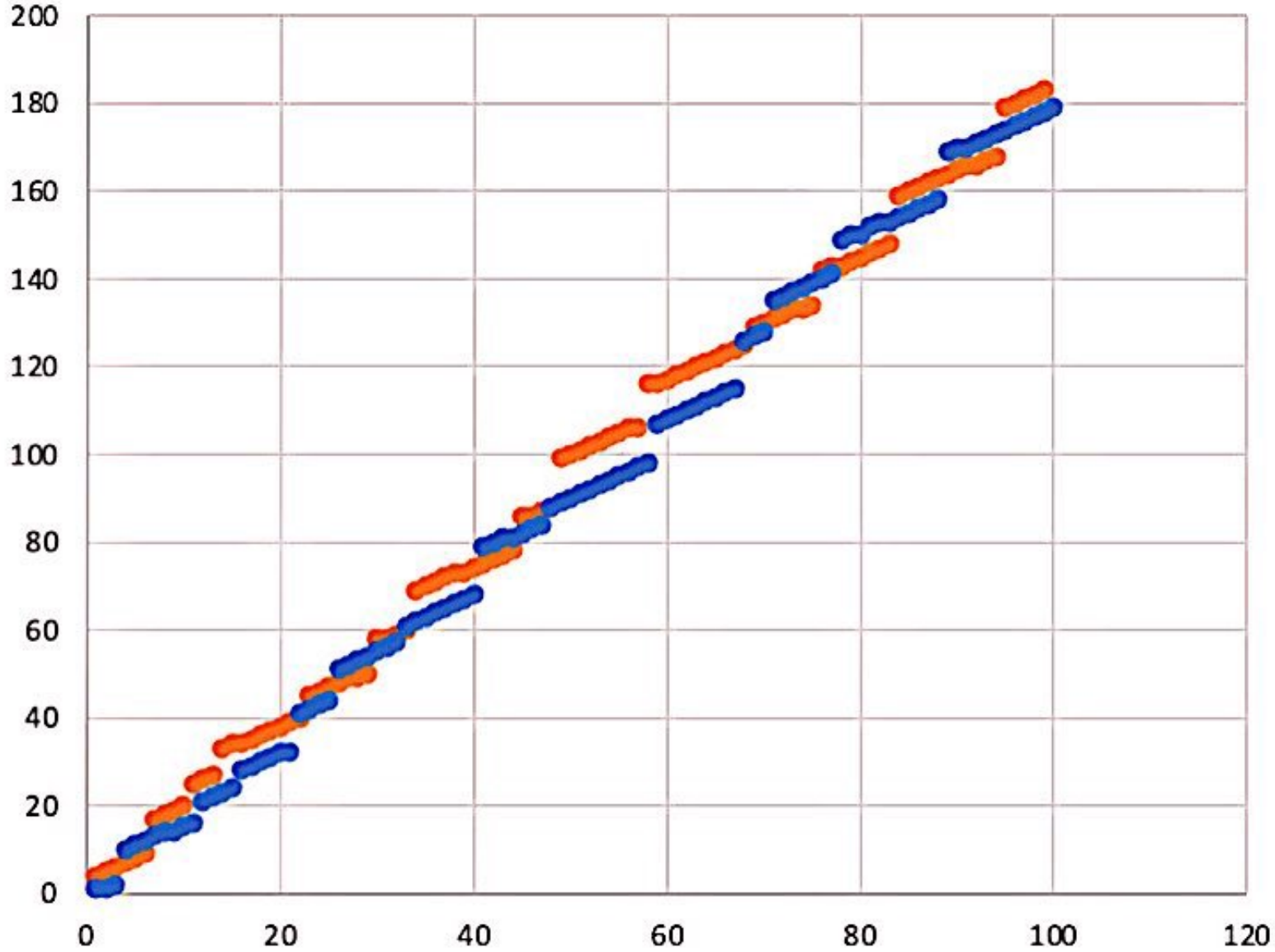
10/20

2018213354 윤비.

1. (1) 실행 4-2. , 프로세스 2개 일 때.



실험 4-2



그림에서는 잘 안 보이지만 2개의 process가 서로 번갈아가며 논리 CPU 사용한다.

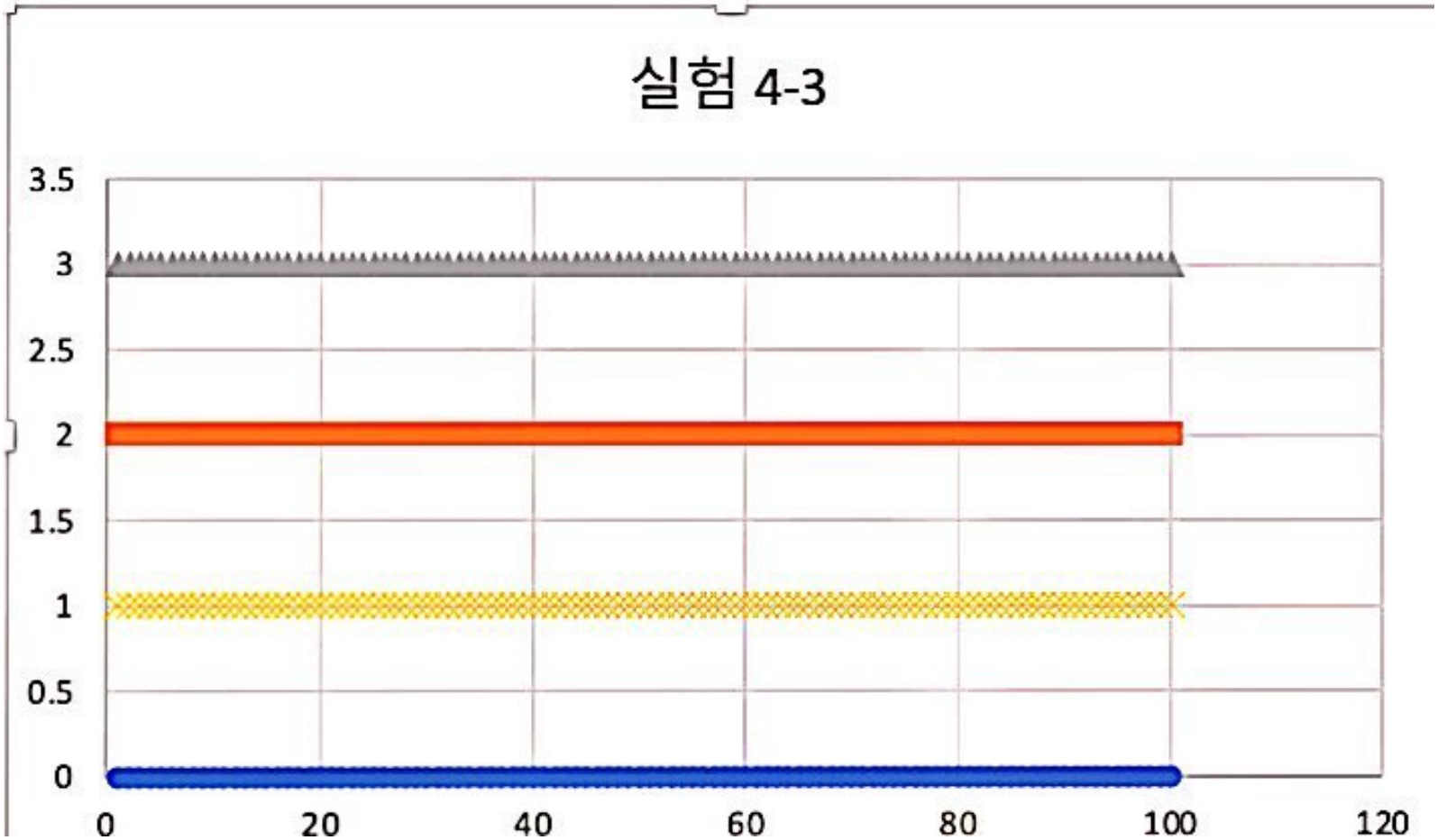
(동시에 사용XX).

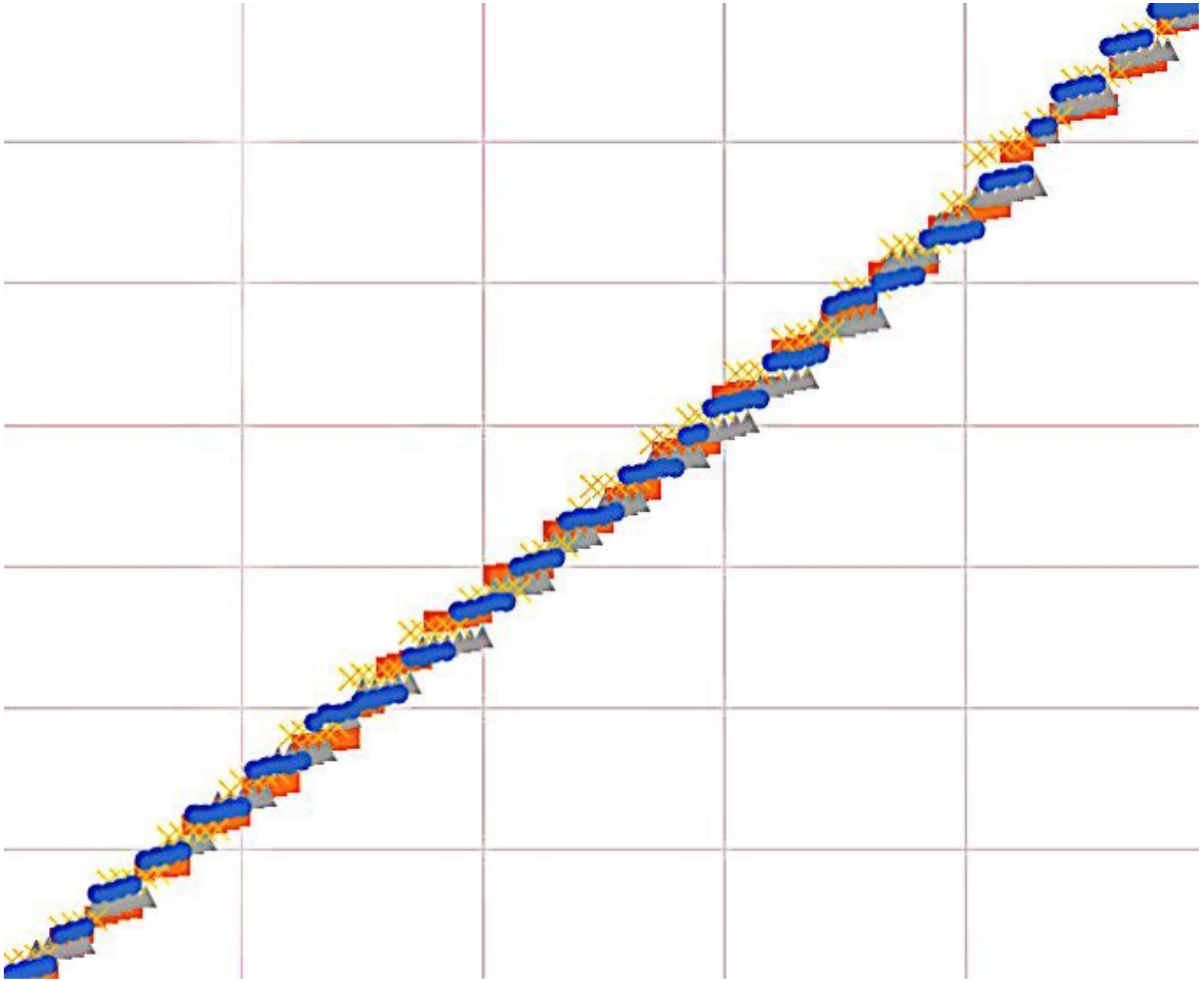
각 process는 논리 CPU를 사용하고 있을 때만 처리가 진행됨을 알 수 있다.

단위 시간별 진행도는 process가 1개일 때에 비교해서 약 절반 정도.

pdf에 있는 실험 4-1 보고 (처리 도중까지) 그래프 참고.

(2) 실습 4-3, 프로세스 추가 및 제.





∴ 동시에 여러 process 실행해도 결국 큐에서 동작하는 process는 그대로??

여러 process가 큐를 순차적으로 그대로 동작한다.

첫번째 process ~ 마지막 process까지 한사이클 돌면 다시 첫번째 process동작

(RR방식)

2. throughput과 latency 측정하기.

- throughput => 단위 시간당 처리된 일의 양을수치 화한 것. (단위: process수/경과시간)
- latency => 처리가 시작 ~ 종료까지 경과된 시간으로 측정수치 화한 것 (처리종료시간 - 처리시작시간)

(1) process 2개일 때

throughput => 2 process / 0.2 [s]
= 10process / 1 [s].

latency = process 0 ~ process 1의 latency
= 200.0 [ms].

(2) process 4개일 때

throughput => 4process / 0.4 [s]
= 10 process / 1 [s]

latency => process 0 ~ process 3의 latency.
= 400 [ms].

3. multiprocessor.

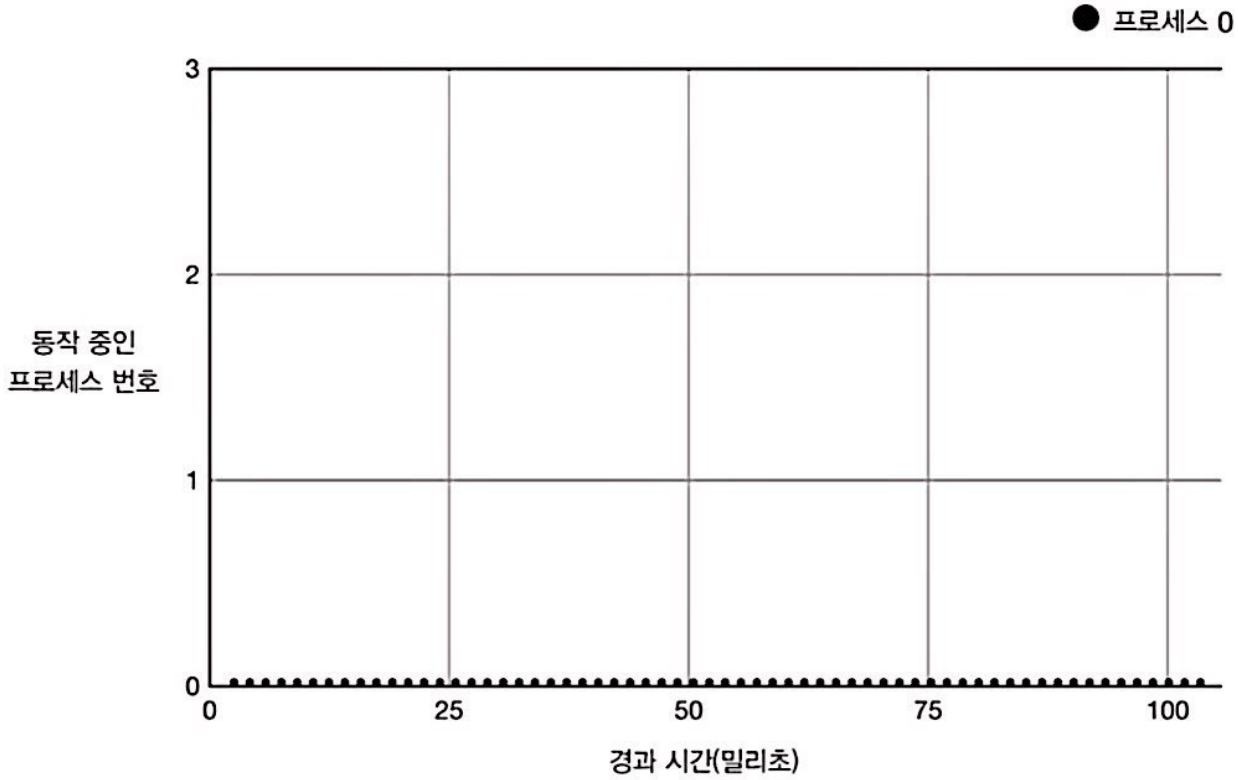
확인해보니 내 system의 논리 cpu 개수는 1개이다.

1. sched 프로그램은 여러 개의 논리 cpu가 인식되는 system에서만 Test 할 수 있는 것 같다. 따라서 이 부분은 다른 교재를 참고하여 더 공부했다. (그 교재에서 보여준 결과를 토대로 분석해보았다.)

프로세스 1개 : 실험 4-D

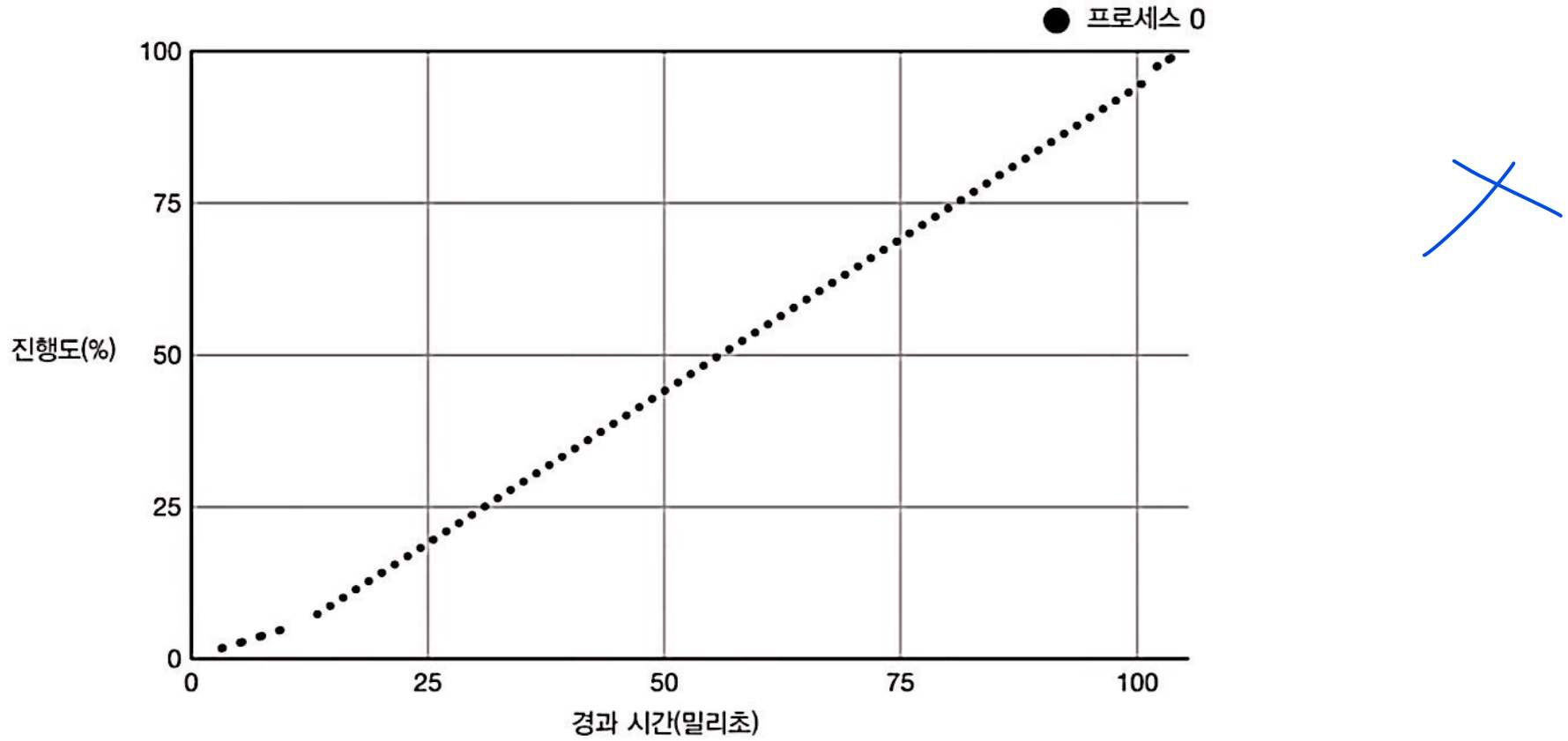
프로세스가 1개인 경우에는 [그림 4-25]와 같이 동작합니다.

그림 4-25 논리 CPU에 동작 중인 프로세스 (실험 4-D, **그래프 ①**)



프로세스 001 시스템마다 논리 CPU에서 계속 동작중이고.
(다른 쪽의 논리 CPU는 idle 상태)

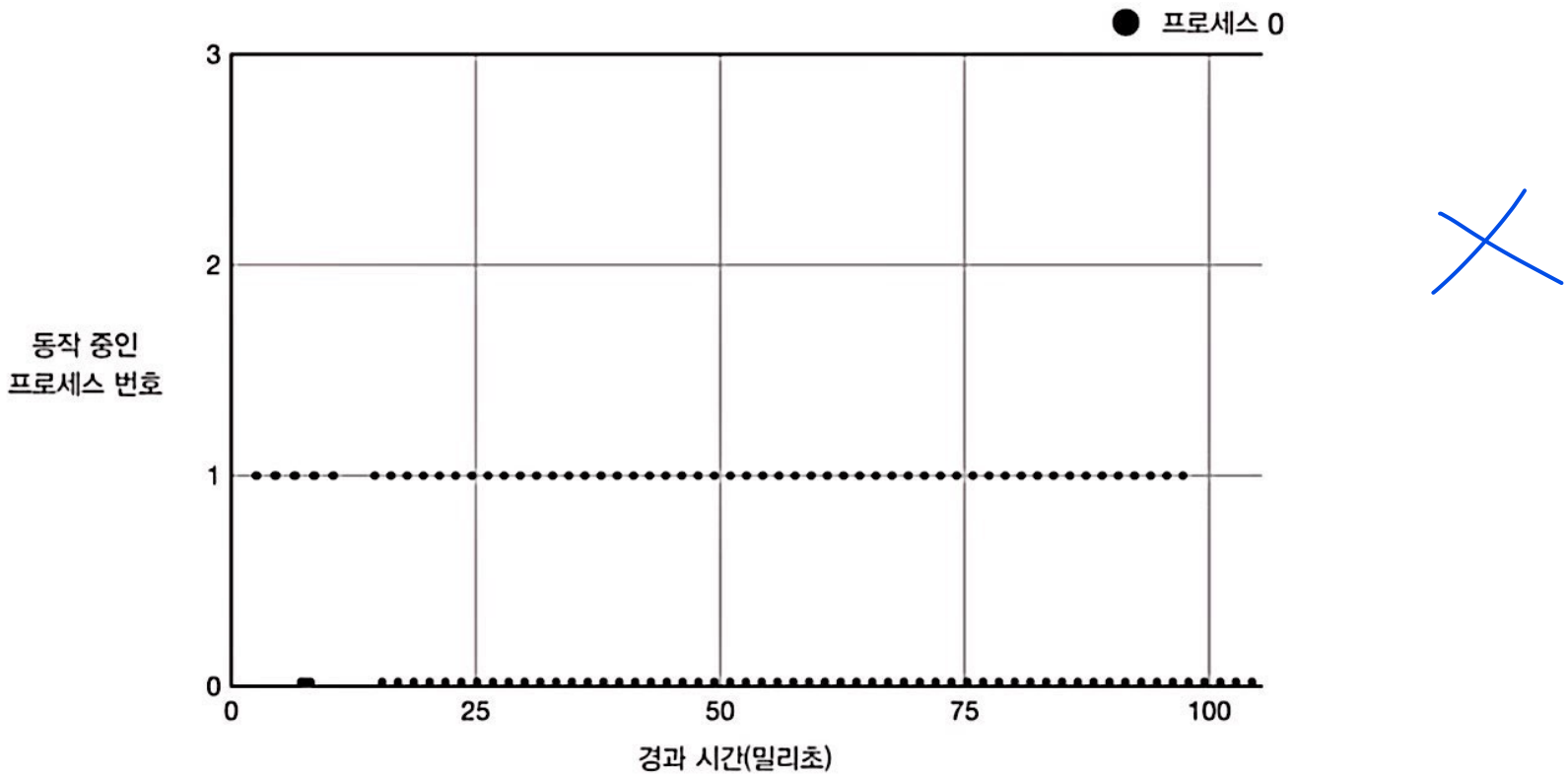
그림 4-26 프로세스 0의 진행도 (실험 4-D, **그래프 ②**)



process의 실행도이다. 논리 CPU가 1개일 때는 같다.

process 2개 일 때는.

그림 4-27 논리 CPU에 동작 중인 프로세스 (실험 4-E, **그래프 ①**)

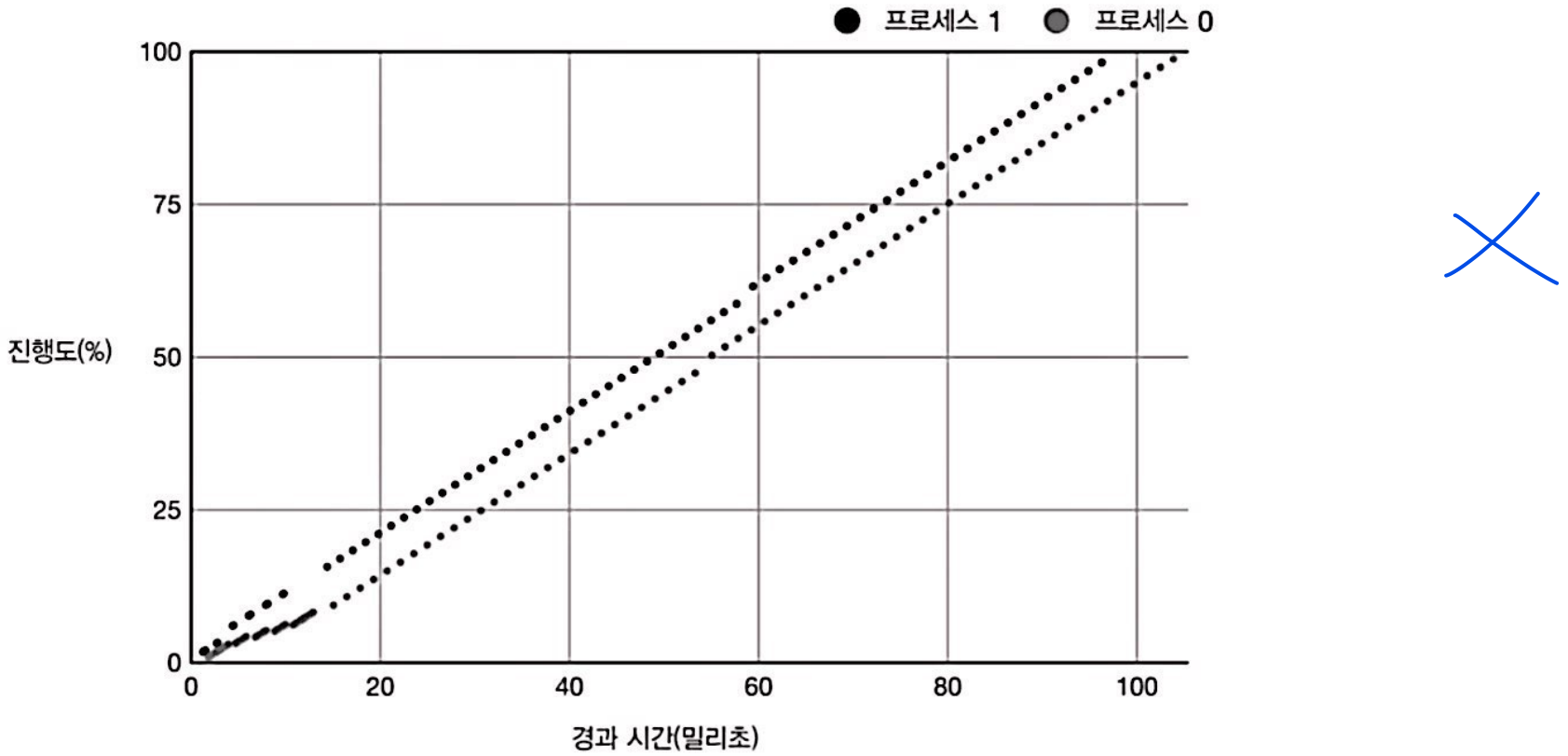


이렇게 동작한다.

각각의 CPU에 process 0~ process 3이 동시에 동작한다.

idle 상태의 CPU가 없으므로 현재 리소스를 최대한 사용하고 있다.

그림 4-28 프로세스 0과 프로세스 1의 진행도 (실험 4-E, **그래프 ②**)

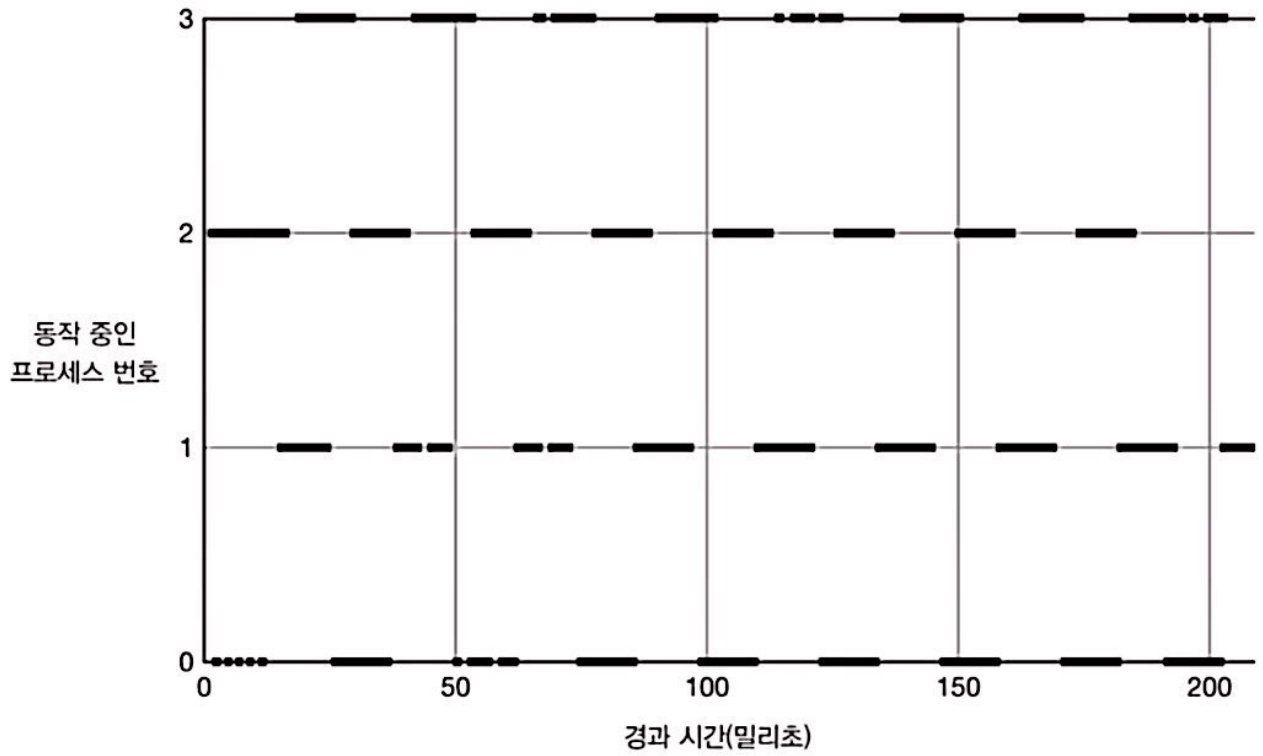


즉 process 진행도이다. 즉 process가 논리 CPU를 독점해서 동작한다.

그래프로 논리 CPU가 1개인 경우에 비해 2번의 시간차가 존재한다.

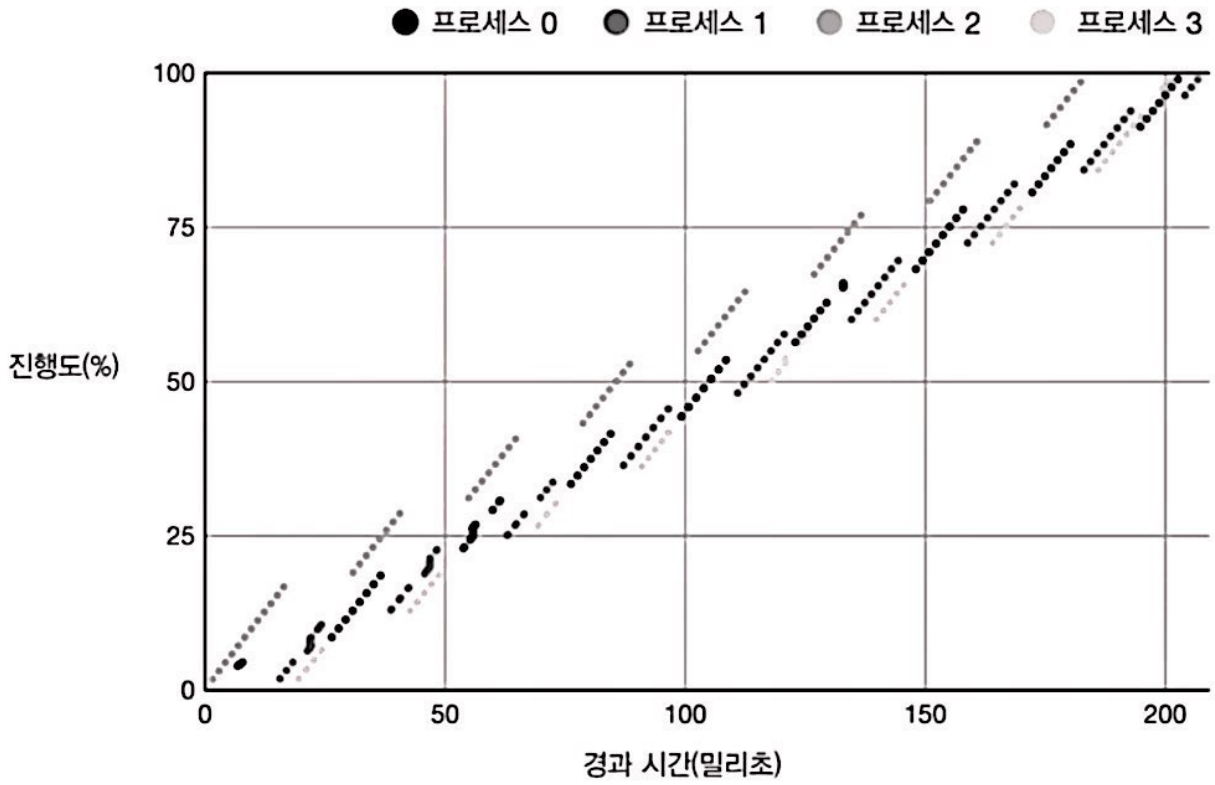
process가 4개인 경우는

그림 4-29 논리 CPU에 동작 중인 프로세스 (실험 4-F, **그래프 ①**)



2개의 코어 cpu에 각각 2개의 process들이 서로 번갈아가며 동작하고있다.

그림 4-30 프로세스 0~프로세스 3의 진행도 (실험 4-F, **그래프 ②**)



process 진행도를 보면 각 process가 거의 동시에 진행되고 있음을 알 수 있다.
최종적으로 각 process가 온전히 cpu를 독점 가능했던 경우에 비해 소요시간 x2 걸렸다.

각 설명에서 throughput과 latency는

→ 설명 4-D

throughput => 1 process / 100[ms]

= 1 process / 0.1[s]

= 10 process / 1[s]

latency => 100[ms]

설명 4-E

throughput => 2 process / 100[ms]

= 2 process / 0.1[s]

= 20 process / 1[s]

latency => 100[ms].

설명 4-F

throughput => 4 process / 200[ms]

= 4 process / 0.2[s]

= 20 process / 1[s]

latency => 200[ms]

```
judith@judith-VirtualBox:~$ time taskset -c 0 ./sched 1 10000 10000
estimating the workload which takes just one milli-second...
end estimation
0       9463    100

real    0m10.954s
user    0m10.457s
sys     0m0.063s
```

```
$ time taskset -c 0 ./sched 2 10000 10000
1 19716 100
0 19732 100

real 0m21.487s
user 0m21.480s
sys 0m0.000s
$
```

경과시간과 사용시간의 차이점은.

전처리 시간을 빼면 경과시간, 사용시간 둘다 거의 같다.

(cpu 점유율이 높으니까).

```
$ time taskset -c 0,1 ./sched 1 10000 10000
0 9813 100

real 0m11.569s
user 0m11.564s
sys 0m0.000s
```

논리 CPU가 1개인 경우에 대한 data가 있음.

C CPU2개라고 한쪽 CPU는 계속 idle 상태(?).

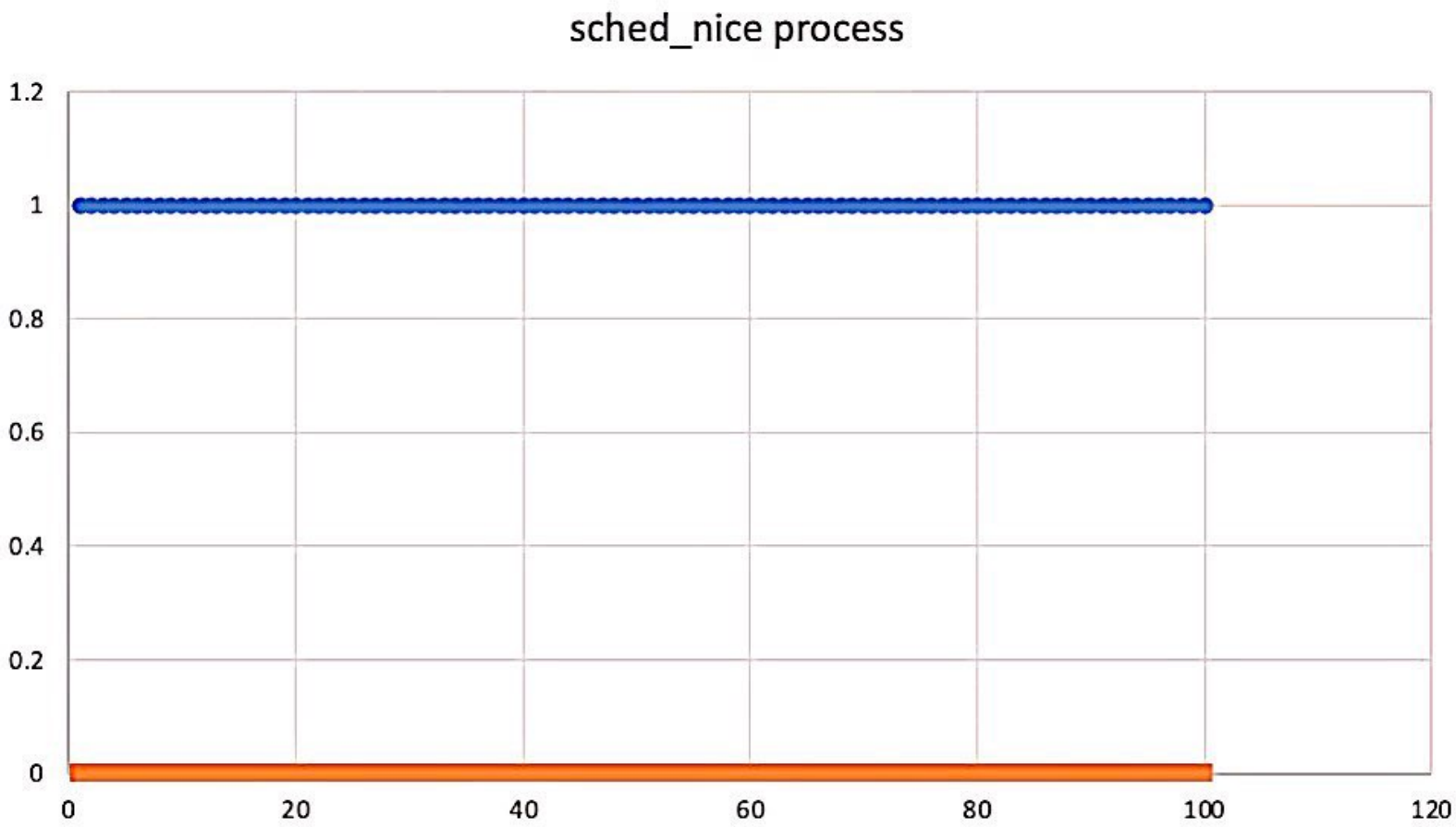
```
$ time taskset -c 0,1 ./sched 2 10000 10000
1 10326 100
0 10350 100

real 0m12.103s
user 0m22.424s
sys 0m0.000s
```

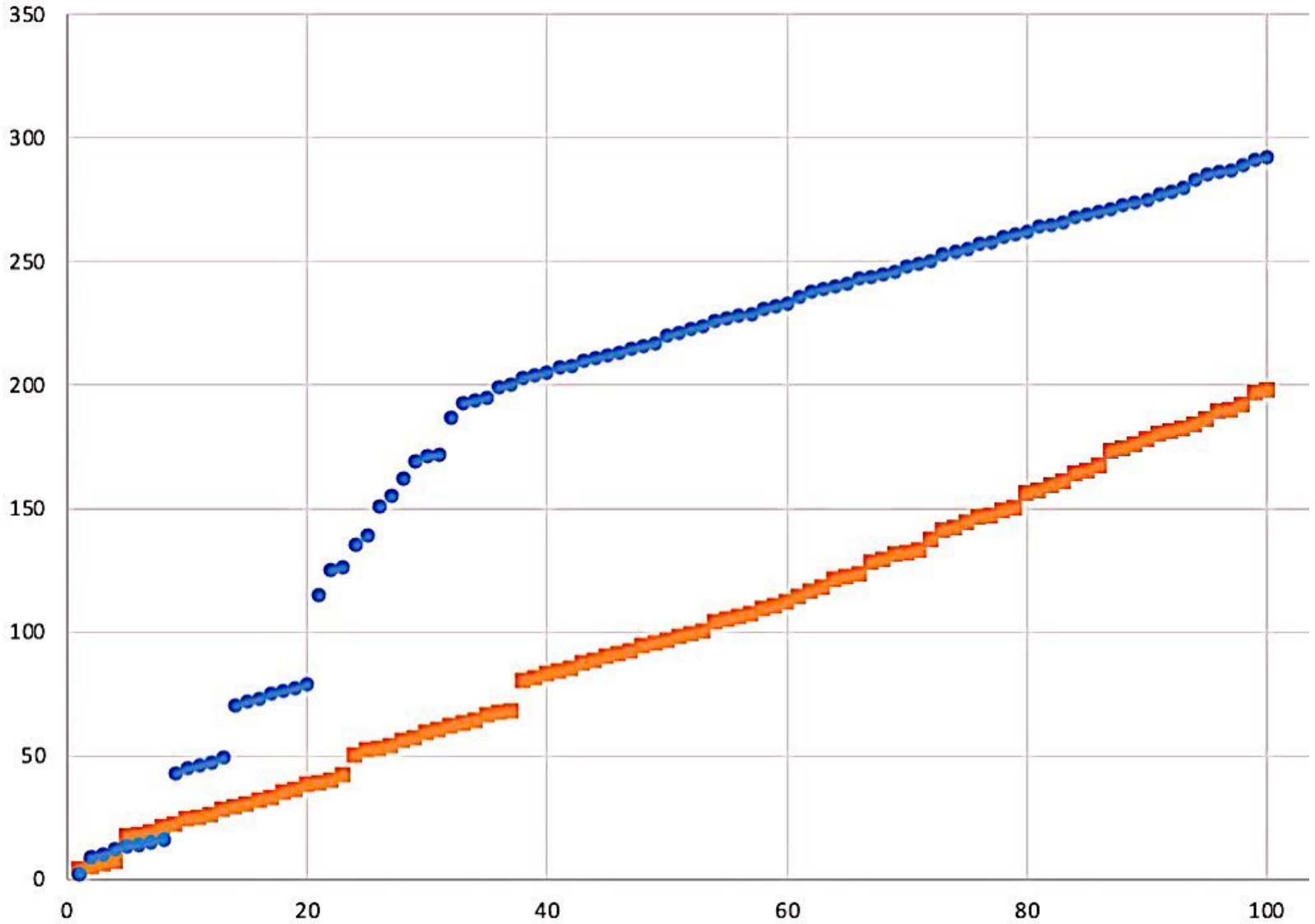
정답시간은 CPU가 1, process가 2일 때의 가장 최적의 사용시간은 2이다.

(따로따로 CPU에 2개의 process 실행 가능한).

(3). nice( ).



너무 출출해서 타는 잘 안나진만 현재는 우선순위가 높은 process이
process와 비교해서 더높은 CPU시간 완료.



process001 우선순위를 내세워 순서 먼저 종료됐을 것 같았는

C process는 그 다음에 종료됨?