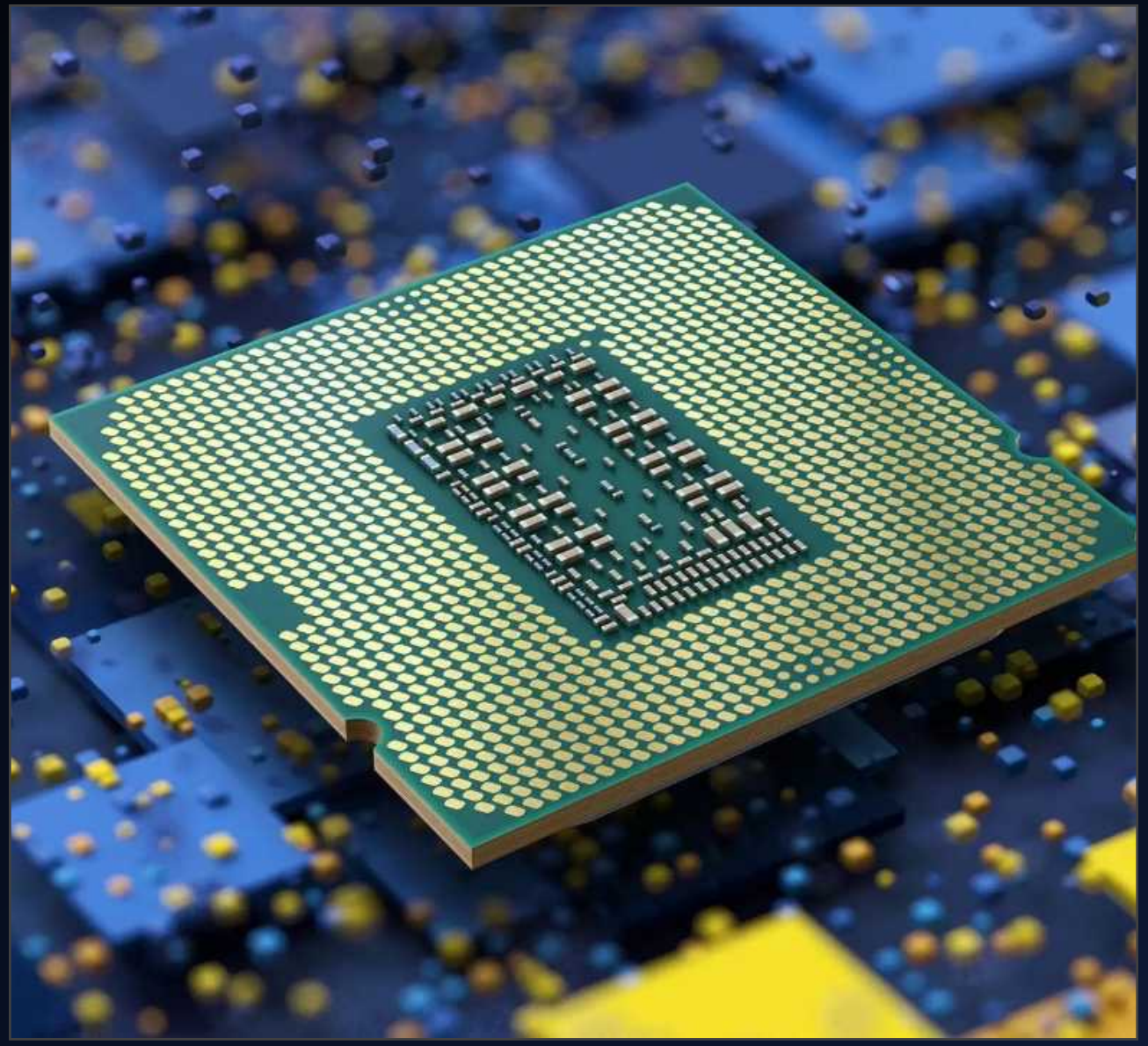
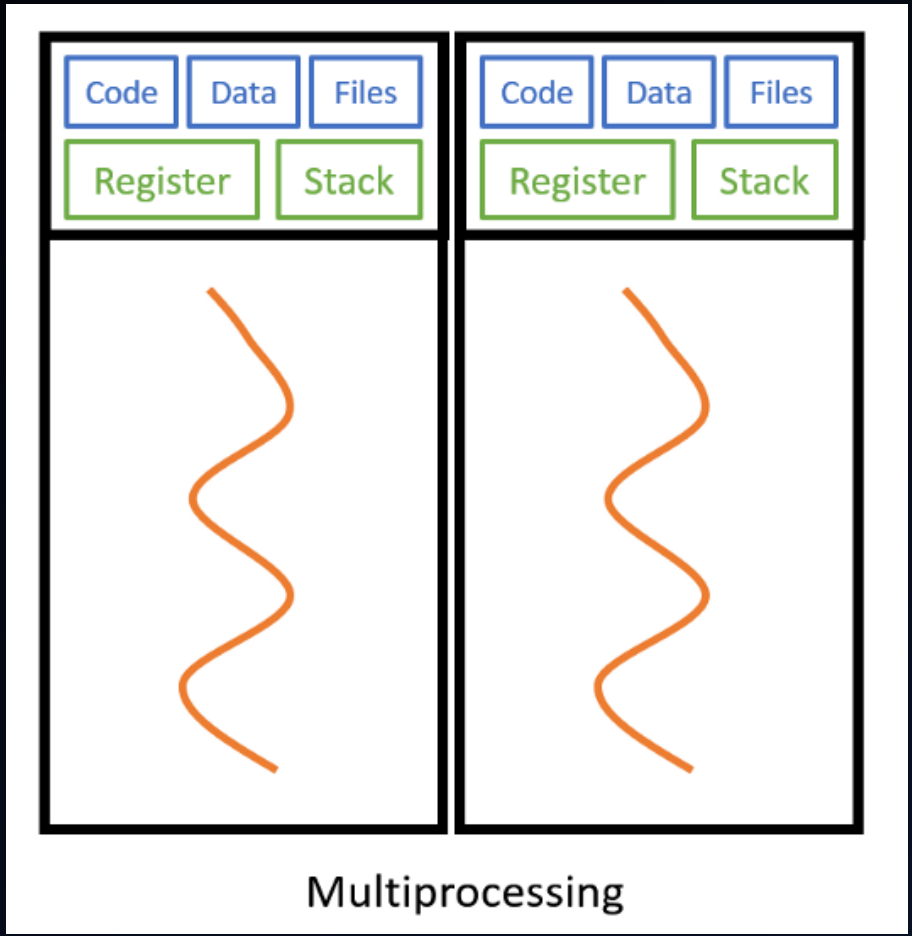
IMPLEMENTATION OF CPU SCHEDULER SIMULATOR



image_src : https://www.pcworld.com/article/541489/how-to-check-cpu-and-memory-usage.html

Process

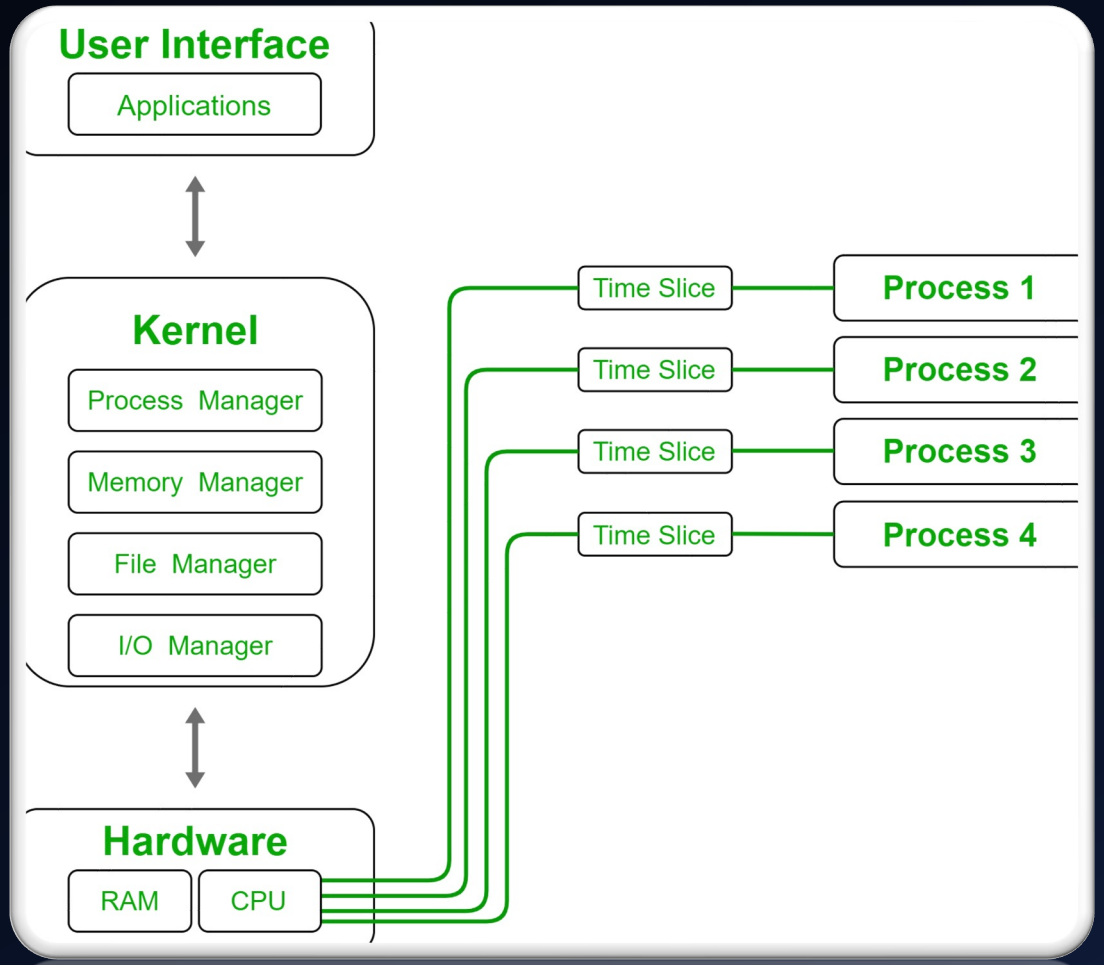


- 컴퓨터에서 실행중인 프로그램을 의미한다
- CPU의 발달로 대부분 멀티 프로세싱을 사용함.

image_src : https://towardsdatascience.com/multithreading-and-multiprocessing-in-10-minutes-20d9b3c6a867

CPU Scheduling

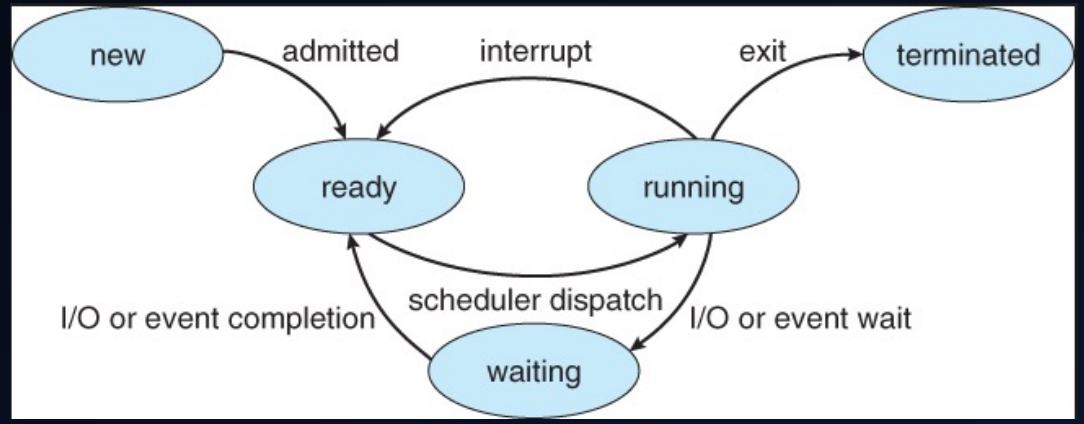
- 프로세스가 작업을 수행할 때, 언제 어떤 프로세스에 CPU를 할당할지를 결정하는 작업이다.



image_src : https://www.geeksforgeeks.org/time-slicing-in-cpu-scheduling/

Process State Diagram

- New
  - 프로세스가 생성된 상태
- Ready
  - 프로세스가 CPU에 할당되기 위한 준비가 완료된 상태
- Running
  - 프로세스가 CPU에 할당되어 실행중인 상태
- Waiting
  - Block 되어 I/O나 event를 기다리는 상태
- Terminated
  - 프로세스가 종료가 된 상태



CPU Scheduling Terminologies

- Burst time(실행시간)
  - 한 프로세스가 실행되는 시간
- Arriving time(도착시간)
  - 한 프로세스가 ready queue에 도착한 시간
- Priority(우선순위)
  - 여러 프로세스들 중 먼저 실행이 될 순서를 뜻함
- Time Quantum
  - 실행의 최소시간 단위

CPU Scheduling Criteria

- Turnaround time (반환시간)
  - 한 프로세스가 Cpu를 점유하기 시작한 순간부터 Cpu점유를 반환한 시점까지의 시간
- Waiting time (대기시간)
  - 프로세스의 Cpu 대기 시간
- Response time (응답시간)
  - 프로세서가 대기큐에 들어와서 처음으로 Cpu점유를 얻게된대까지 걸리는 시간

CPU Scheduling Algorithms

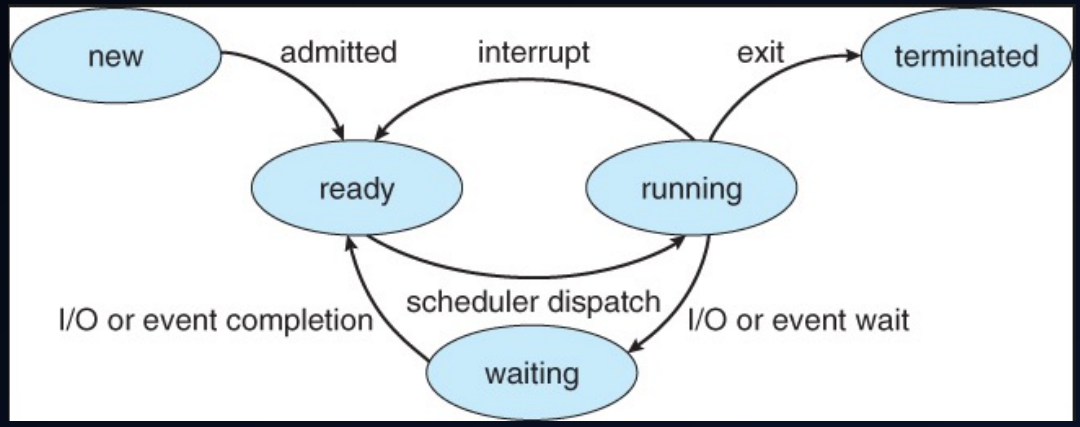
- FCFS(First Come, First Served)
- SJF(Shortest job first)
- SRTF(Shortest Remaining Time First)
- PS(Priority Scheduling)
- RR(Round Robin)
- HRN(Highest Response Ratio Next)

CPU Scheduler 시뮬레이터 설계

- 프로그램 실행 방법
  - {algorithm} {process cores} -b {n} {n} ... -a {n} {n} ... -pr {n} {n} ... -t {n} {n} ...
- 초기화 단계
  - Option Parsing 및 초기화
  - Process Table 구현 및 초기화
- Algorithm 함수 실행 단계
  - 프로세스 개수 만큼 프로세스 fork()
  - Option에 입력된 값에 따라 스케줄링 알고리즘 실행
    - FCFS / SJF / SRTF / PS / RR / HRN 알고리즘

CPU Scheduler 설계 - 스케줄링 알고리즘 실행 (여러 공통함수들)

- void arriving_wait(t_data *data, t_PCB *pcb, uint64_t start, int id)
  - 각 프로세스의 옵션에 입력된 arriving time에 맞게 각 프로세스를 지연시켜 ready queue로 보냄
  - srcs/wait.c 참고
- void dispatcher(t_PCB *pcb)
  - Ready queue에 있던 프로세스가 running 상태로 전이
  - race condition을 막기위해 sem_wait으로 다른 프로세스들을 막아둠.
  - srcs/dispatcher.c 참고



- void termination(t_PCB *pcb)
  - Burst time을 다한 프로세스를 종료 시킴
  - 다음 프로세스가 실행될 수 있도록 sem_post를 통해 block을 풂
  - srcs/termination.c 참고

image_src : https://www.toppr.com/ask/question/cpu-scheduling/

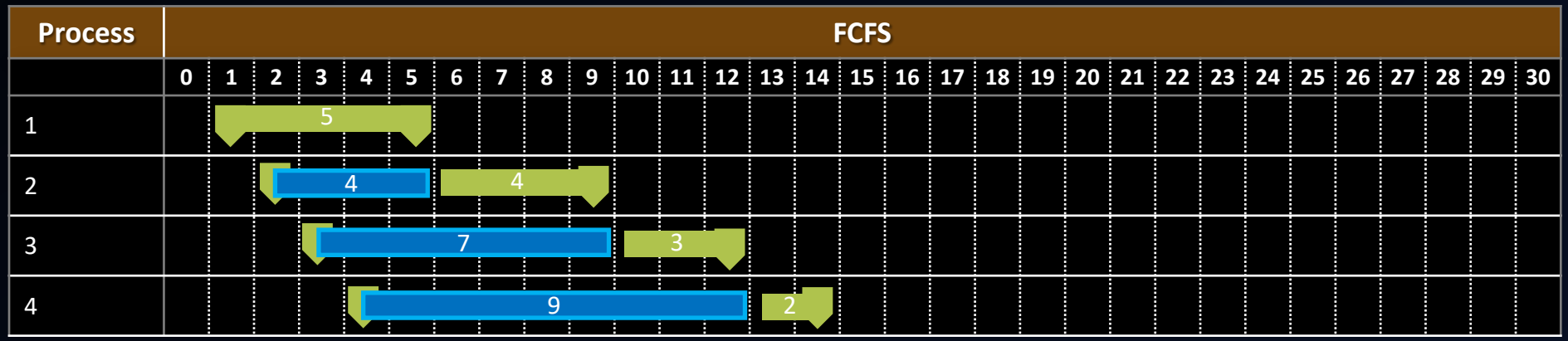
CPU SCHEDULER 설계 - 스케줄링 알고리즘 실행 (FCFS)

- FCFS함수
  - start_process 함수
    - 프로세스 생성
    - 각 프로세스당 FCFS_start 함수 호출
- FCFS_start 함수
  - Arriving_wait 함수
    - Arriving time만큼 기다림
  - Dispatcher 함수
    - 먼저 도착한 프로세스를 running 상태로 보냄
  - FCFS_running 함수
    - Burst time 만큼 실행
  - Termination 함수
    - 프로세스 종료(다음 프로세스 실행가능하게 만듬)

```
void    FCFS_running(t_PCB *pcb)
{
    while (1)
    {
        update_cost_time(pcb);
        if (pcb->cost_time > pcb->data->burst_time[pcb->user_id])
            break ;
        pcb->resister += pcb->cost_time;
    }
}

int FCFS_start(t_data *data, t_process_table_node *process_table_node)
{
    t_PCB *pcb;

    pcb = process_table_node->pcb;
    arriving_wait(data, pcb, pcb->process_start, pcb->user_id);
    pcb->state = READY;
    dispatcher(pcb);
    FCFS_running(pcb);
    termination(pcb);
    return (0);
}

int FCFS(t_data *data)
{
    start_process(data);
    sem_post(data->wait);
    return (0);
}
```

FCFS

FCFS 4 -b 5 4 3 2 -a 1 2 3 4



waiting running

```
alvinlee ~/Git_Folder/cpu-scheduler-simulater  main ±
 ./cpu-scheduler FCFS 4 -b 5 4 3 2 -a 1 2 3 4

####################          cpu scheduling information          ####################

PROCESS 1 :     burst_time      5000  arriving_time     1000  priority  0  time_quantum 0
PROCESS 2 :     burst_time      4000  arriving_time     2000  priority  0  time_quantum 0
PROCESS 3 :     burst_time      3000  arriving_time     3000  priority  0  time_quantum 0
PROCESS 4 :     burst_time      2000  arriving_time     4000  priority  0  time_quantum 0


######################                    result                    ######################

PROCESS: 1              PID: 99022              Turnaround time: 5001           Response time: 0                Waiting time: 0
PROCESS: 2              PID: 99023              Turnaround time: 8002           Response time: 4001             Waiting time: 4001
PROCESS: 3              PID: 99024              Turnaround time: 10003          Response time: 7002             Waiting time: 7002
PROCESS: 4              PID: 99025              Turnaround time: 11004          Response time: 9003             Waiting time: 9003
alvinlee ~/Git_Folder/cpu-scheduler-simulater  main ±
```

CPU SCHEDULER 설계 - 스케줄링 알고리즘 실행 (SJF{1})

- SJF함수
  - Sort함수
    - Shortest job을 sorting 해줌
- SJF_start 함수
  - SJF_wait 함수
    - Shortest job으로 sorting된 값들 순서대로 프로세스 running 상태로 보낼 준비

```
void    SJF_running(t_PCB *pcb)
{
    while (1)
    {
        update_cost_time(pcb);
        if (pcb->cost_time > pcb->data->burst_time[pcb->user_id])
            break ;
        pcb->resister += pcb->cost_time;
    }
}

int SJF_start(t_data *data, t_process_table_node *process_table_node)
{
    t_PCB *pcb;

    pcb = process_table_node->pcb;
    arriving_wait(data, pcb, pcb->process_start, pcb->user_id);
    SJF_wait(data, pcb, pcb->user_id);
    dispatcher(pcb);
    SJF_running(pcb);
    termination(pcb);
    return (0);
}

int SJF(t_data *data)
{
    sort(data, data->process_table, "SJF");
    start_process(data);
    return (0);
}
```

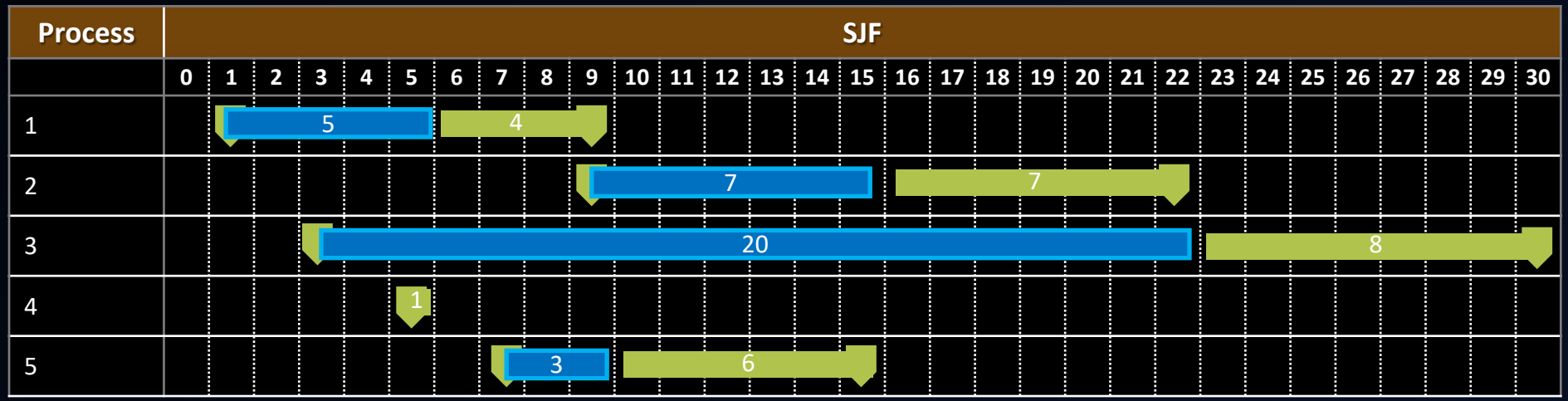
CPU Scheduler 설계 - 스케줄링 알고리즘 실행 (SJF{2})

- Sort 함수에서 shortest job(shortest burst time)순으로 prioirty에다가 순서를 sorting해둠
- 따라서 priority 순으로 sem_wait을 시키고 순서대로 dispatcher로 보내는 것이 가능

```
void    SJF_wait(t_data *data, t_PCB *pcb, int id)
{
    pcb->state = WAITING;
    while (1)
    {
        sem_wait(data->moniter_wait);
        if (data->priority[id] == (uint64_t)data->done)
        {
            data->done += 1;
            sem_wait(data->wait);
            break ;
        }
        sem_post(data->moniter_wait);
    }
    pcb->state = READY;
    sem_post(data->wait);
    sem_post(data->moniter_wait);
}
```

SJF

SJF 5 -b 4 7 8 1 6 -a 1 9 3 5 7



waiting running

```
alvinlee ~/Git_Folder/cpu-scheduler-simulater  main ±
 ./cpu-scheduler SJF 5 -b 4 7 8 1 6 -a 1 9 3 5 7


####################           cpu scheduling information             ####################


PROCESS 1 :     burst_time      4000  arriving_time     1000  priority  0  time_quantum 0
PROCESS 2 :     burst_time      7000  arriving_time     9000  priority  0  time_quantum 0
PROCESS 3 :     burst_time      8000  arriving_time     3000  priority  0  time_quantum 0
PROCESS 4 :     burst_time      1000  arriving_time     5000  priority  0  time_quantum 0
PROCESS 5 :     burst_time      6000  arriving_time     7000  priority  0  time_quantum 0



######################                        result                        ######################


PROCESS: 1              PID: 97463              Turnaround time: 9005           Response time: 5004             Waiting time: 5004
PROCESS: 2              PID: 97464              Turnaround time: 14006          Response time: 7005             Waiting time: 7005
PROCESS: 3              PID: 97465              Turnaround time: 28007          Response time: 20006            Waiting time: 20006
PROCESS: 4              PID: 97466              Turnaround time: 1001           Response time: 0                Waiting time: 0
PROCESS: 5              PID: 97467              Turnaround time: 9005           Response time: 3004             Waiting time: 3004
alvinlee ~/Git_Folder/cpu-scheduler-simulater  main ±
```

CPU SCHEDULER 설계 - 스케줄링 알고리즘 실행 (SRTF{1})

- SRTF_start 함수
  - pthread_create/pthread_detach
    - Comp_moniter함수를 서브스레드로 실행시킴.
  - SRTF_wait 함수
    - Comp_moniter 함수에서 프로세스의 상태를 Running 상태로 바꾸면 dispatch될 수 있도록 함.
  - waiting_zone함수
    - 더 짧은 remaining time인 프로세스가 나타나서 running이 되어버리면 Running중이였던 프로세스는 waiting상태로 들어가고 waiting_zone에 갇히게 됨.
    - Waiting이 풀리면(해당 프로세스가 더 짧은 remaining time이면) 모니터에서 data->done을 해당 process id로 바꿔줌. 따라서 break함.

```
void    SRTF_wait(t_PCB *pcb)
{
    while (1)
    {
        if (pcb->state == RUNNING)
        {
            sem_post(pcb->data->moniter_wait);
            sem_wait(pcb->data->wait);
            break ;
        }
    }
}
```

```
void    waiting_zone(t_data *data, int id)
{
    while (1)
    {
        if (id == data->done)
        {
            data->done = -1;
            break ;
        }
    }
}
```

```
void    SRTF_running(t_PCB *pcb)
{
    while (1)
    {
        update_cost_time(pcb);
        if (pcb->state == WAITING)
        {
            pcb->wait_start = get_time();
            sem_post(pcb->data->moniter_wait);
            sem_post(pcb->data->dispatcher);
            waiting_zone(pcb->data, pcb->user_id);
            sem_wait(pcb->data->dispatcher);
            pcb->wait_start = get_time() - pcb->wait_start;
            pcb->waiting_time += pcb->wait_start;
            pcb->running_start = get_time() - pcb->cost_time;
        }
        if (pcb->cost_time > pcb->data->burst_time[pcb->user_id])
            break ;
        pcb->resister += pcb->cost_time;
    }
}

int SRTF_start(t_data *data, t_process_table_node *process_table_node)
{
    t_PCB *pcb;

    pcb = process_table_node->pcb;
    pthread_create(&pcb->tid1, NULL, comp_moniter, (void *)pcb);
    pthread_detach(pcb->tid1);
    arriving_wait(data, pcb, pcb->process_start, pcb->user_id);
    SRTF_wait(pcb);
    dispatcher(pcb);
    sem_post(pcb->data->wait);
    SRTF_running(pcb);
    termination(pcb);
    return (0);
}

int SRTF(t_data *data)
{
    start_process(data);
    return (0);
}
```

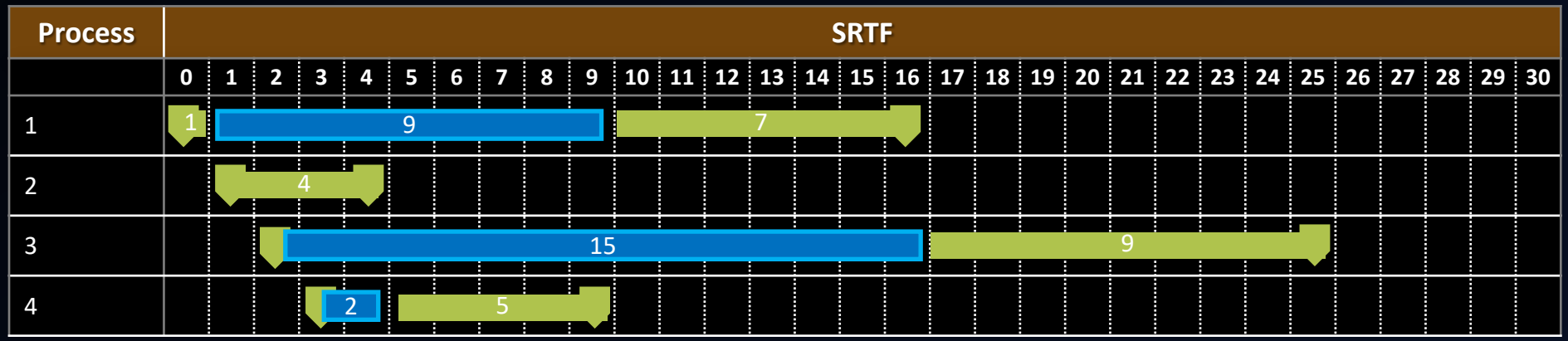
CPU SCHEDULER 설계 - 스케줄링 알고리즘 실행 (SRTF{2})

- comp_moniter 함수
  - While(1)로 계속해서 끊임없이 실행
  - If (running)
    - 지금 실행중인 프로세스와 다른프로세스들의 remaining time을 비교하여 더 짧은 프로세스가 나타나면 해당 프로세스를 running 상태로 바꿔줌.
  - If (!running && !new)
    - 실행중이 아닌 ready상태인 프로세스가 SRTF_wait을 나와 처음 실행되도록 하기위해
    - Terminate된 프로세스 때문에 위의 if(running)조건에 해당되는 프로세스가 전혀 없어진 경우 다음 shortest remaining 프로세스를 실행하기 위해

```
void    *comp_moniter(void *pcb_v)
{
    t_PCB   *pcb;
    t_PCB   *pcb_rec;
    t_process_table_node    *process_table_node;
    uint64_t    temp;
    int flag;
    int i;

    pcb = (t_PCB *)pcb_v;
    process_table_node = pcb->data->process_table->head->next;
    while (1)
    {
        sem_wait(pcb->data->moniter_wait);
        if (pcb->state == RUNNING)
        {
            i = -1;
            pcb_rec = pcb;
            temp = pcb->burst_time - pcb->cost_time;
            process_table_node = pcb->data->process_table->head->next;
            while (++i < pcb->data->process_cores)
            {
                if (pcb->user_id != i && (process_table_node->pcb->state == READY || \
                    process_table_node->pcb->state == WAITING) \
                    && process_table_node->pcb->burst_time - \
                    process_table_node->pcb->cost_time <= temp)
                {
                    temp = process_table_node->pcb->burst_time - \
                            process_table_node->pcb->cost_time;
                    pcb_rec = process_table_node->pcb;
                }
                process_table_node = process_table_node->next;
                usleep(10);
            }
            if (pcb_rec != pcb)
            {
                pcb_rec->state = RUNNING;
                pcb->data->done = pcb_rec->user_id;
                pcb->state = WAITING;
            }
        }
        if (pcb->state != RUNNING && pcb->state != NEW)
        {
            i = -1;
            flag = 0;
            pcb_rec = pcb;
            temp = pcb->burst_time - pcb->cost_time;
            process_table_node = pcb->data->process_table->head->next;
            while (++i < pcb->data->process_cores)
            {
                if (process_table_node->pcb->state != TERMINATED && process_table_node->pcb->state != NEW && process_table_node->pcb->burst_time - \
                    process_table_node->pcb->cost_time <= temp)
                {
                    temp = process_table_node->pcb->burst_time - \
                            process_table_node->pcb->cost_time;
                    pcb_rec = process_table_node->pcb;
                    flag = 1;
                }
                process_table_node = process_table_node->next;
                usleep(10);
            }
            if (flag)
            {
                pcb_rec->state = RUNNING;
                pcb->data->done = pcb_rec->user_id;
                if (pcb->state == READY)
                    pcb->state = WAITING;
            }
        }
        sem_post(pcb->data->moniter_wait);
    }
    return ((void *)0);
}
```

SRTF

SRTF 4 -a 0 1 2 3 -b 8 4 9 5



waiting running

```
alvinlee ~/Git_Folder/cpu-scheduler-simulater  main ±
./cpu-scheduler SRTF 4 -a 0 1 2 3 -b 8 4 9 5


####################            cpu scheduling information            ####################


PROCESS 1 :     burst_time      8000  arriving_time     0  priority     0  time_quantum 0
PROCESS 2 :     burst_time      4000  arriving_time     1000  priority  0  time_quantum 0
PROCESS 3 :     burst_time      9000  arriving_time     2000  priority  0  time_quantum 0
PROCESS 4 :     burst_time      5000  arriving_time     3000  priority  0  time_quantum 0


######################                        result                        ######################


PROCESS: 1              PID: 762              Turnaround time: 17003          Response time: 0              Waiting time: 9002
PROCESS: 2              PID: 763              Turnaround time: 4001           Response time: 0              Waiting time: 0
PROCESS: 3              PID: 764              Turnaround time: 24004          Response time: 15003          Waiting time: 15003
PROCESS: 4              PID: 765              Turnaround time: 7000           Response time: 1999           Waiting time: 1999
alvinlee ~/Git_Folder/cpu-scheduler-simulater  main ±
```

CPU SCHEDULER 설계 - 스케줄링 알고리즘 실행 (PS)

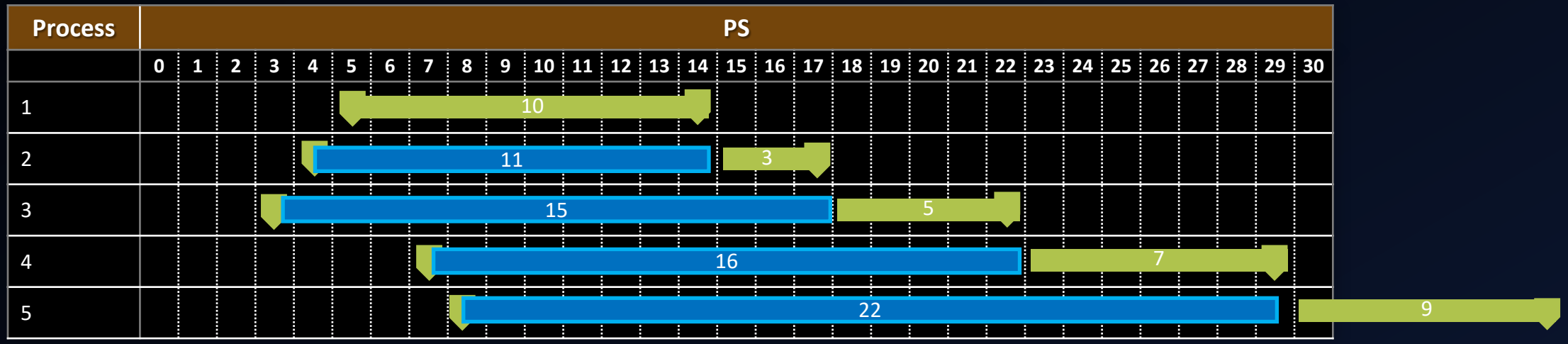
- PS함수
  - Data->done을 0으로 초기화
  - (priority 0부터 실행을 위하여)
- PS_start 함수
  - PS_wait 함수
    - Priority에 맞게 순서대로 실행이 되도록 함.

```
void    PS_wait(t_data *data, t_PCB *pcb, int id)
{
    pcb->state = WAITING;
    while (1)
    {
        sem_wait(data->moniter_wait);
        if (data->priority[id] == (uint64_t)data->done)
        {
            data->done += 1;
            sem_wait(data->wait);
            break ;
        }
        sem_post(data->moniter_wait);
    }
    pcb->state = READY;
    sem_post(data->wait);
    sem_post(data->moniter_wait);
}
```

```
void    PS_running(t_PCB *pcb)
{
    while (1)
    {
        update_cost_time(pcb);
        if (pcb->cost_time > pcb->data->burst_time[pcb->user_id])
            break ;
        pcb->resister += pcb->cost_time;
    }
}

int PS_start(t_data *data, t_process_table_node *process_table_node)
{
    t_PCB *pcb;

    pcb = process_table_node->pcb;
    arriving_wait(data, pcb, pcb->process_start, pcb->user_id);
    PS_wait(data, pcb, pcb->user_id);
    dispatcher(pcb);
    PS_running(pcb);
    termination(pcb);
    return (0);
}

int PS(t_data *data)
{
    data->done = 0;
    start_process(data);
    return (0);
}
```

PS

PS 5 -pr 0 1 2 3 4 -a 5 4 3 7 8 -b 10 3 5 7 9



waiting running

```
alvinlee ~/Git_Folder/cpu-scheduler-simulater  main ±
./cpu-scheduler PS 5 -pr 0 1 2 3 4 -a 5 4 3 7 8 -b 10 3 5 7 9

####################          cpu scheduling information          ####################

PROCESS 1 :     burst_time      10000  arriving_time    5000  priority  0  time_quantum 0
PROCESS 2 :     burst_time      3000  arriving_time     4000  priority  1  time_quantum 0
PROCESS 3 :     burst_time      5000  arriving_time     3000  priority  2  time_quantum 0
PROCESS 4 :     burst_time      7000  arriving_time     7000  priority  3  time_quantum 0
PROCESS 5 :     burst_time      9000  arriving_time     8000  priority  4  time_quantum 0


######################                    result                    ######################

PROCESS: 1              PID: 3721               Turnaround time: 10001          Response time: 0                Waiting time: 0
PROCESS: 2              PID: 3722               Turnaround time: 14002          Response time: 11001            Waiting time: 11001
PROCESS: 3              PID: 3723               Turnaround time: 20003          Response time: 15002            Waiting time: 15002
PROCESS: 4              PID: 3724               Turnaround time: 23003          Response time: 16002            Waiting time: 16002
PROCESS: 5              PID: 3725               Turnaround time: 31005          Response time: 22004            Waiting time: 22004
alvinlee ~/Git_Folder/cpu-scheduler-simulater  main ±
```

CPU SCHEDULER 설계 - 스케줄링 알고리즘 실행 (RR{1})

```c
void	waiting_zone(t_data *data, int id)
{
	while (1)
	{
		if (id == data->done)
		{
			data->done = -1;
			break ;
		}
	}
}
```

- RR 함수
  - Sort함수
    - RR의 실행기준은 arrived 한 순서로 실행함.
- RR_start 함수
  - pthread_create/pthread_detach
    - Timequantum_moniter함수를 서브스레드로 실행시킴.

```c
void	RR_running(t_PCB *pcb)
{
	while (1)
	{
		update_cost_time(pcb);
		if (pcb->state == WAITING)
		{
			pcb->wait_start = get_time();
			sem_post(pcb->data->moniter_wait);
			sem_post(pcb->data->dispatcher);
			waiting_zone(pcb->data, pcb->user_id);
			sem_wait(pcb->data->dispatcher);
			pcb->wait_start = get_time() - pcb->wait_start;
			pcb->waiting_time += pcb->wait_start;
			pcb->running_start = get_time() - pcb->cost_time;
		}
		if (pcb->cost_time > pcb->data->burst_time[pcb->user_id])
			break ;
		pcb->resister += pcb->cost_time;
	}
}

int RR_start(t_data *data, t_process_table_node *process_table_node)
{
	t_PCB *pcb;

	pcb = process_table_node->pcb;
	pthread_create(&pcb->tid1, NULL, timequantum_moniter, (void *)pcb);
	pthread_detach(pcb->tid1);
	arriving_wait(data, pcb, pcb->process_start, pcb->user_id);
	dispatcher(pcb);
	RR_running(pcb);
	termination(pcb);

	return (0);
}

int RR(t_data *data)
{
	sort(data, data->process_table, "RR");
	start_process(data);
	return (0);
}
```

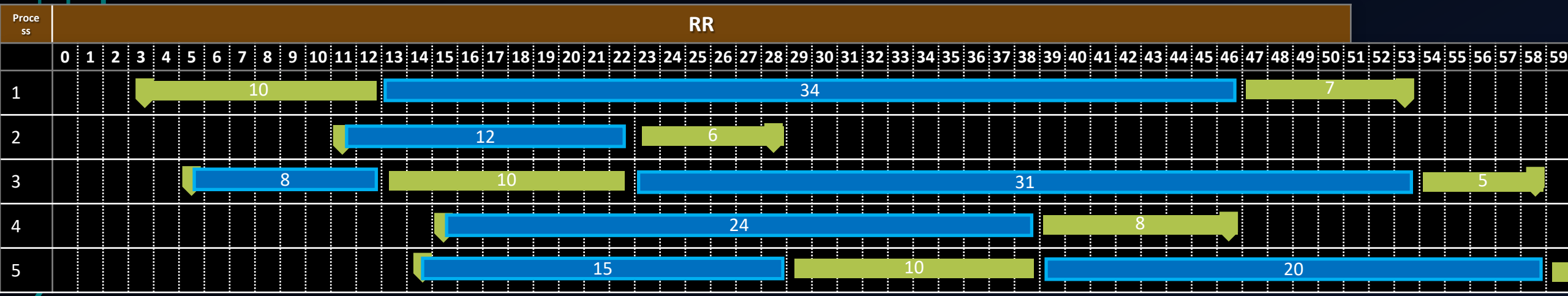
CPU SCHEDULER 설계 - 스케줄링 알고리즘 실행 (RR{2})

- timequantum_moniter 함수
  - While(1)로 계속해서 끊임없이 실행
  - If (running)
    - 지금 실행중인 프로세스가 timequantum만큼 실행이 되었으면 바로 다음 실행 순서대로 다음 프로세스를 선점을 해버림.
  - If (!running && !new && !terminated)
    - Terminate된 프로세스 때문에 위의 if(running)조건에 해당되는 프로세스가 전혀 없어진 경우 다음 우선순위대로 실행을 시키기 위해

```
void    *timequantum_moniter(void *pcb_v)
{
    t_PCB *pcb;
    uint64_t    priority;
    t_process_table_node    *process_table_node;
    int i, j, flag_1;

    pcb = (t_PCB *)pcb_v;
    while (1)
    {
        sem_wait(pcb->data->moniter_wait);
        if (pcb->state == RUNNING)
        {
            if (pcb->cost_time / pcb->time_quantum > (uint64_t)pcb->repeated_times \
                && pcb->cost_time % pcb->time_quantum <= 500)
            {
                pcb->repeated_times += 1;
                priority = -1;
                i = -1;
                flag_1 = 0;
                priority = pcb->data->priority[pcb->user_id];
                while (++i < pcb->data->process_cores)
                {
                    j = -1;
                    priority++;
                    if (priority == (uint64_t)pcb->data->process_cores)
                        priority = 0;
                    process_table_node = pcb->data->process_table->head->next;
                    while (++j < pcb->data->process_cores)
                    {
                        if (priority == pcb->data->priority[j] && \
                            (process_table_node->pcb->state == WAITING || \
                                process_table_node->pcb->state == READY))
                        {
                            flag_1 = 1;
                            process_table_node->pcb->state = RUNNING;
                            pcb->data->done = process_table_node->pcb->user_id;
                            pcb->state = WAITING;
                            break ;
                        }
                        process_table_node = process_table_node->next;
                        usleep(10);
                    }
                    if (flag_1 == 1)
                        break ;
                }
            }
        }
        else if (pcb->state != RUNNING && pcb->state != NEW && pcb->state != TERMINATED)
        {
            i = -1;
            flag_1 = 0;
            process_table_node = pcb->data->process_table->head->next;
            sem_wait(pcb->data->wait);
            if (pcb->data->terminated != -1 && pcb->data->terminated != pcb->data->last_terminated)
            {
                pcb->data->last_terminated = pcb->data->terminated;
                priority = -1;
                i = -1;
                priority = pcb->data->priority[pcb->data->terminated];
                while (++i < pcb->data->process_cores)
                {
                    j = -1;
                    priority++;
                    if (priority == (uint64_t)pcb->data->process_cores)
                        priority = 0;
                    process_table_node = pcb->data->process_table->head->next;
                    while (++j < pcb->data->process_cores)
                    {
                        if (priority == process_table_node->pcb->data->priority[j] && \
                            (process_table_node->pcb->state == WAITING || \
                                process_table_node->pcb->state == READY))
                        {
                            flag_1 = 1;
                            process_table_node->pcb->state = RUNNING;
                            pcb->data->done = process_table_node->pcb->user_id;
                            break ;
                        }
                        process_table_node = process_table_node->next;
                        usleep(10);
                    }
                    if (flag_1 == 1)
                        break ;
                }
            }
            sem_post(pcb->data->wait);
        }
        sem_post(pcb->data->moniter_wait);
    }
}
```

RR

RR 5 -b 17 6 15 8 13 -a 3 11 5 15 14 -t 10 10 10 10 10



waiting running

```
alvinlee ~/Git_Folder/cpu-scheduler-simulater  main ±
./cpu-scheduler RR 5 -b 17 6 15 8 13 -a 3 11 5 15 14 -t 10 10 10 10 10


####################           cpu scheduling information             ####################


PROCESS 1 :     burst_time      17000  arriving_time    3000  priority  0  time_quantum 10000
PROCESS 2 :     burst_time      6000  arriving_time     11000  priority  0  time_quantum 10000
PROCESS 3 :     burst_time      15000  arriving_time    5000  priority  0  time_quantum 10000
PROCESS 4 :     burst_time      8000  arriving_time     15000  priority  0  time_quantum 10000
PROCESS 5 :     burst_time      13000  arriving_time    14000  priority  0  time_quantum 10000



######################                          result                          ######################


PROCESS: 1              PID: 5589               Turnaround time: 51007          Response time: 0                Waiting time: 34006
PROCESS: 2              PID: 5590               Turnaround time: 18001          Response time: 12000            Waiting time: 12000
PROCESS: 3              PID: 5591               Turnaround time: 54008          Response time: 8000             Waiting time: 39007
PROCESS: 4              PID: 5592               Turnaround time: 32005          Response time: 24004            Waiting time: 24004
PROCESS: 5              PID: 5593               Turnaround time: 48008          Response time: 15003            Waiting time: 35007
alvinlee ~/Git_Folder/cpu-scheduler-simulater  main ±
```

CPU SCHEDULER 설계 - 스케줄링 알고리즘 실행 (HRN{1})

```c
void    HRN_wait(t_data *data, t_PCB *pcb, int id)
{
    pcb->state = WAITING;
    while (1)
    {
        sem_wait(data->stop);
        if (id == data->done)
        {
            sem_post(pcb->data->moniter_wait);
            sem_wait(data->wait);
            sem_post(data->stop);
            break ;
        }
        sem_post(data->stop);
    }
    pcb->state = READY;
    sem_post(data->wait);
    sem_post(data->stop);
}
```

- HRN_start 함수
  - pthread_create/pthread_detach
    - Response_rate_moniter함수를 서브스레드로 실행시킴.
  - HRN_wait함수
    - Response_rate_moniter 함수에서 프로세스의 상태를 Running 상태로 바꾸면 dispatch될 수 있도록 함.

- Response rate
  - (대기시간 + burst time) / burst time

```c
void    HRN_running(t_PCB *pcb)
{
    while (1)
    {
        update_cost_time(pcb);
        if (pcb->cost_time > pcb->data->burst_time[pcb->user_id])
            break ;
        pcb->resister += pcb->cost_time;
    }
}

int HRN_start(t_data *data, t_process_table_node *process_table_node)
{
    t_PCB *pcb;

    pcb = process_table_node->pcb;
    pthread_create(&pcb->tid1, NULL, response_rate_moniter, (void *)pcb);
    pthread_detach(pcb->tid1);
    arriving_wait(data, pcb, pcb->process_start, pcb->user_id);
    HRN_wait(data, pcb, pcb->user_id);
    dispatcher(pcb);
    HRN_running(pcb);
    termination(pcb);
    return (0);
}

int HRN(t_data *data)
{
    start_process(data);
    return (0);
}
```

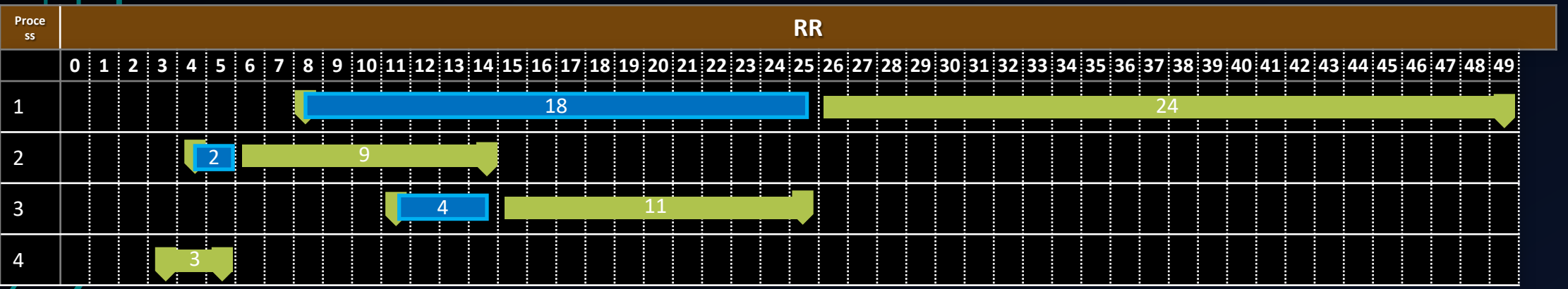
CPU Scheduler 설계 - 스케줄링 알고리즘 실행 (HRN{2})

```
void    *response_rate_moniter(void *pcb_v)
{
    t_PCB    *pcb, *pcb_rec_1, *pcb_rec_2;
    t_process_table_node    *process_table_node;
    long double response_rate, temp;
    int i, flag_1, flag_2;

    pcb = (t_PCB *)pcb_v;
    while (1)
    {
        sem_wait(pcb->data->moniter_wait);
        if (pcb->state == WAITING && pcb->data->terminated != pcb->data->last_terminated)
        {
            i = -1;
            flag_1 = 0;
            flag_2 = 0;
            response_rate = 0;
            temp = 1;
            process_table_node = pcb->data->process_table->head->next;
            while (++i < pcb->data->process_cores)
            {
                if (process_table_node->pcb->state == WAITING || process_table_node->pcb->state == RUNNING)
                {
                    pcb->data->last_terminated = pcb->data->terminated;
                    response_rate = (((long double)get_time() - (long double)process_table_node->pcb->readyque_arrived_time) + \
                                    (long double)process_table_node->pcb->burst_time) / (long double)process_table_node->pcb->burst_time;
                    pcb_rec_2 = process_table_node->pcb;
                    if (response_rate > temp)
                    {
                        pcb_rec_1 = process_table_node->pcb;
                        temp = response_rate;
                        flag_1 = 1;
                    }
                }
                else
                {
                    flag_2++;
                }
                process_table_node = process_table_node->next;
            }
            if (flag_1)
                pcb->data->done = pcb_rec_1->user_id;
            else if (flag_2 == pcb->data->process_cores - 1)
                pcb->data->done = pcb_rec_2->user_id;
        }
        else if (pcb->state == WAITING && pcb->data->terminated == -1)
            pcb->data->done = pcb->user_id;
        sem_post(pcb->data->moniter_wait);
    }
}
```

- Response_rate_moniter 함수
  - While(1)로 계속해서 끊임없이 실행
  - If (waiting && terminated != last_terminated)
    - Waiting이고 한 프로세스가 terminated 된 순간에(→HRN은 선점형이 아니라서 하나가 끝나야 실행되기 위해) 모든 running or waiting 프로세스에 대해서 response_rate를 계산하고, 가장 큰 프로세스로 바꿈.

HRN

HRN 4 -b 24 9 11 3 -a 8 4 11 3



waiting running

```
alvinlee ~/Git_Folder/cpu-scheduler-simulater  main ±
 ./cpu-scheduler HRN 4 -b 24 9 11 3 -a 8 4 11 3


####################           cpu scheduling information             ####################


PROCESS 1 :     burst_time      24000  arriving_time    8000  priority  0  time_quantum 0
PROCESS 2 :     burst_time      9000  arriving_time     4000  priority  0  time_quantum 0
PROCESS 3 :     burst_time      11000  arriving_time    11000  priority 0  time_quantum 0
PROCESS 4 :     burst_time      3000  arriving_time     3000  priority  0  time_quantum 0


#####################                       result                       #####################


PROCESS: 1              PID: 8582               Turnaround time: 42004          Response time: 18003            Waiting time: 18003
PROCESS: 2              PID: 8583               Turnaround time: 11002          Response time: 2001             Waiting time: 2001
PROCESS: 3              PID: 8584               Turnaround time: 15003          Response time: 4002             Waiting time: 4002
PROCESS: 4              PID: 8585               Turnaround time: 3001           Response time: 0                Waiting time: 0
alvinlee ~/Git_Folder/cpu-scheduler-simulater  main ±
```