Term Project (CPU Scheduling Simulator) 보고서

컴퓨터학과 2021xxxxxx 박찬우

개요

CPU 스케줄러(Scheduler)는 운영체제의 핵심 구성 요소 중 하나로, 준비 상태(Ready State)에 있는 여러 프로세스 중 어떤 프로세스에게 CPU를 할당할 것인지 결정하는 역할을 한다. CPU는 컴퓨터 시스템에서 가장 중요한 자원 중 하나이므로, 이를 효율적으로 사용하는 것이 시스템 전체 성능에 큰 영향을 미친다. CPU 스케줄러는 프로세스의 도착 시간(Arrival time), CPU 처리 시간(CPU Burst time), 대기 시간 (Waiting time), 반환 시간 (Turnaround time) 등을 최소화하는 것을 목표로 한다.

본 프로젝트에서는 C언어를 사용하여 다양한 CPU Scheduling algorithm들을 시뮬레이션하는 프로그램을 구현하였다. 구현된 스케줄러는 다음과 같다:

1. FCFS (First-Come, First-Served): 가장 간단한 Scheduling algorithm으로, ready queue에 도착한 순서대로 프로세스를 처리한다. 비선점형 (Non-preemption) 방식이다.

2. SJF (Shortest Job First): CPU 처리 시간(CPU Burst time)이 가장 짧은 프로세스에게 CPU를 먼저 할당한다.
   • Non-Preemption SJF (비선점형 SJF): 일단 CPU를 할당 받으면 해당 처리 시간(Burst time)이 끝날 때까지 실행된다.
   • Preemption SJF (선점형 SJF): 현재 실행 중인 프로세스보다 잔여 실행 시간이 더 짧은 프로세스가 도착하면 CPU를 빼앗아 할당한다. 평균 대기 시간(Average Waiting time)을 최소화하는데 최적이다.

3. Priority Scheduling (우선순위 스케줄링): 각 프로세스에 우선순위를 부여하고, 가장 높은 우선순위를 가진 프로세스에게 CPU를 할당한다.
   • Non-Preemption Priority (비선점형 우선순위): 일단 CPU를 할당 받으면 해당 처리 시간이 끝날 때까지 실행된다.
   • Preemption Priority (선점형 우선순위): 현재 실행 중인 프로세스보다 우선순위가 높은 프로세스가 ready queue에 도착하면 CPU를 빼앗아 할당한다.

4. RR (Round Robin): 각 프로세스에 동일한 크기의 Time Quantum (시간 할당량)을 부여하여, 할당된 시간만큼 실행한 후 ready queue 맨 뒤로 이동한다.

본 시뮬레이터는 프로세스의 도착 시간, CPU 처리 시간(CPU Burst Time), I/O 발생 시각(I/O Request Time), I/O 처리 시간(I/O Burst Time)을 고려하여 각 알고리즘을 동작한

다. 실행 후에는 간트 차트(Gantt Chart), 평균 대기 시간(Average Waiting Time), 평균 반환 시간(Average Turnaround Time)등의 성능 평가 지표를 제공하여 알고리즘 간 비교 분석을 가능하게 한다.

CPU 스케줄링 알고리즘의 동작을 이해하고 성능을 분석하기 위해 다양한 시뮬레이터들이 개발되어 있다. 이러한 시뮬레이터들은 GUI 환경을 제공하거나, 다양한 파라미터 설정을 통해 폭넓은 실험을 가능하게 한다.

• CPUSim: CPU 스케줄링 알고리즘 교육의 어려움을 해결하기 위해 JAVA와 Swing 언어 기반으로 개발된 교육용 시뮬레이터이다. CPU 스케줄링 알고리즘(FCFS, HRN, SJF, SRT, 우선순위, 라운드 로빈)의 동작을 시각적으로 보여준다. 해당 시뮬레이터도 알고리즘 처리 과정을 간트 차트로 보여주고, 평균 대기시간과 평균 반환시간과 같은 성능 수치를 표시한다.

• Web-based Scheduling Visualizer: 웹 브라우저 상에서 스케줄링 알고리즘과 프로세스의 도착 시간과 처리 시간을 정의하고, 스케줄링 결과를 표로 확인할 수 있다.[1]

이외에도 수많은 대학 및 연구기관에서 교육 및 연구 목적으로 자체적인 시뮬레이터를 개발하여 사용하고 있다. 이들 대부분은 알고리즘의 시각화, 통계 데이터 수집, 파라미터 튜닝 기능 등을 공통적으로 제공한다.

본론

본 시뮬레이터의 시스템 구성은 그림 1과 같이 개념적으로 표현할 수 있다.

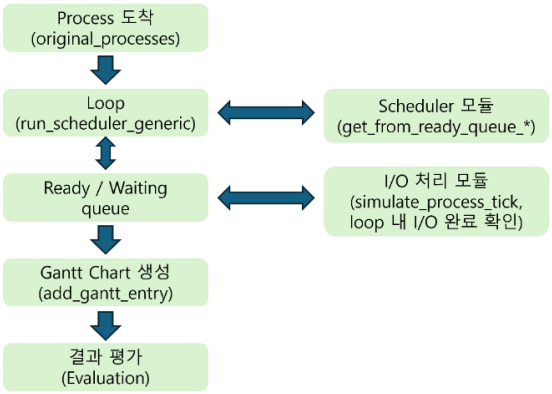

그림 1. 시뮬레이터 시스템 구성

[1] CPU SCHEDULING VISUALIZER . (n.d.). https://cpu-scheduling-visualizer-three.vercel.app/.

기본적인 이벤트 기반 시뮬레이션 구조로, 프로세스(process) – 메인 Loop – Ready/ Waiting queue – Gantt Chart 생성/기록 – 결과 평가 순으로 진행된다.

• 프로세스(Process): 지정된 수의 프로세스를 생성하며, 각 프로세스의 PID, 도착 시간, CPU 처리 시간, I/O 요청 시점, I/O 처리 시간, 우선 순위 등을 무작위로 설정한다.

• 메인 Loop: 입력된 프로세스 데이터를 기반으로 핵심 시뮬레이션을 담당한다.

  • 도착 프로세스 확인: 현재 시간에 도착한 프로세스를 ready queue에 이동시킨다.

  • I/O 완료 확인: I/O 작업이 완료된 프로세스를 I/O Waiting queue에서 Ready queue로 이동시킨다.

  • Ready Queue: CPU 할당을 기다리는 프로세스들의 큐이다. 각 스케줄링 알고리즘에 따라 프로세스를 선택하는 방식이 다르다.

  • CPU: 선택된 프로세스를 실행한다.

  • 프로세스 상태 업데이트: CPU 사용, I/O 발생, 작업 완료 등의 상태 변화를 처리한다.

  • I/O Waiting Queue: I/O 처리 함수와 상호작용하는데, CPU 실행 중 I/O가 발생하면 프로세스는 Waiting queue로 이동한다.

  • Preemption 로직: Preemption 알고리즘의 경우, 현재 실행 중인 프로세스를 중단시키고 다른 프로세스에게 CPU를 할당할지 결정한다.

• Gantt Chart 생성: 메인 loop에서 CPU가 할당되고 해제될 때마다 해당 정보가 이 모듈로 전달되어 간트 차트에 기록된다.

• 결과 평가(Evaluation): 모든 시뮬레이션이 완료된 후, 생성된 간트 차트 정보와 각 프로세스의 최종 상태 정보를 바탕으로 평균 대기 시간, 평균 반환 시간 등의 성능 지표를 계산하여 출력한다.

주요 함수 및 모듈의 알고리즘은 다음과 같다.

1. Create_Process() (프로세스 생성):
   사용자 입력에 따라 난수 기반으로 pid, arrival_time, cpu_burst_time_initial, io_request_time, io_burst_time, priority를 생성한다.
2. reset_process_for_simulation() (시뮬레이션용 프로세스 초기화):
   시뮬레이션 전 각 프로세스의 동적 필드를 초기화하고, 도착 시간을 기준으로 정렬

한다.

3. run_scheduler_generic (스케줄러 공통 반복문)

   아래에 해당 모듈의 논리적 구조를 표현하였다.

   while (completed_count < num_processes):

   1. 도착 프로세스 -> Ready queue
   2. I/O 완료 프로세스 -> Ready queue
   3. Preemption 조건 검사 (Preemptive SJF / Priority)
   4. CPU에 프로세스 할당
   5. 한 tick 실행 (simulate_process_tick)
   6. 종료 또는 I/O queue 대기 처리
   7. 시간 증가 또는 대기 시간 건너뛰기

   end while

   반복문을 빠져나오는 기타 조건:

   1. 현재 시간이 매우 커지는 경우 (5000 단위 시간(tick) 초과)
   2. 모든 프로세스가 완료되었고 CPU가 유휴 상태(Idle)이며, 모든 queue가 비는 경우
   3. 아직 완료되지 않은 프로세스가 있지만, CPU가 유휴 상태이고 모든 큐가 비는 경우. 이 경우에는 다음 이벤트 (새 프로세스 도착 또는 I/O 완료) 시간까지 현재 시간을 점프시키고 간트 차트에 유휴 시간을 기록한다.

   반복문을 빠져나온 뒤에는 Evaluation 함수를 호출한다.

4. 스케줄링 알고리즘별 함수

   get_from_ready_queue_fcfs(): FCFS 알고리즘에 따라 동작한다.

   1. Ready queue가 비었으면 null process를 반환한다.
   2. Ready queue에서 도착 시간이 가장 이른 프로세스를 찾아서 인덱스를 저장한다
   3. 해당 인덱스의 프로세스를 실행하기 위해 ready queue에서 제거한다.

   get_from_ready_queue_sjf(): SJF 알고리즘에 따라 동작한다.

   1. Ready queue가 비었으면 null process를 반환한다.
   2. Ready queue에서 잔여 burst time이 가장 작은 프로세스를 찾는다.
   3. 만약 잔여 burst time이 동일하다면, 도착 시간이 빠른 프로세스를 선택한다.

4. 해당 인덱스의 프로세스를 실행하기 위해 ready queue에서 제거한다.

peek_ready_queue_sjf(): 현재 CPU에서 실행 중인 프로세스와 ready queue에 있는 가장 적합한 후보 프로세스를 비교하기 위해 사용된다. get_from_ready_queue_sjf() 함수와 논리 구조는 유사하다. 후보 프로세스를 ready queue에서 제거하지 않고 상태만 확인하여 preemption 여부를 결정한다.

get_from_ready_queue_priority(): Priority 알고리즘에 따라 동작한다.

1. Ready queue가 비었으면 null process를 반환한다.
2. Ready queue에서 priority 값이 가장 작은 (즉, 우선순위가 가장 높은) 프로세스를 찾는다.
3. 만약 priority가 같다면, 도착 시간이 빠른 프로세스를 선택한다.
4. 해당 인덱스의 프로세스를 실행하기 위해 ready queue에서 제거한다.

peek_ready_queue_priority(): 현재 CPU에서 실행 중인 프로세스와 ready queue에 있는 가장 적합한 후보 프로세스를 비교하기 위해 사용된다. get_from_ready_queue_priority() 함수와 논리 구조는 유사하다. 후보 프로세스를 ready queue에서 제거하지 않고 상태만 확인하여 preemption 여부를 결정한다.

get_from_ready_queue_rr(): Round Robin (Time Quantum=4): Round Robin 알고리즘에 따라 동작한다.

1. Ready queue가 비었으면 null process를 반환한다.
2. Ready queue의 첫번째 (index 0) 프로세스를 실행하기 위해 ready queue에서 제거한다.

5. simulate_process_tick:
개별 프로세스를 단위 시간(tick)만큼 실행하며 다음 역할을 수행한다. 이 함수를 통해 스케줄러 반복문에서 CPU 실행, I/O 전환, 프로세스 종료를 일관되게 다룰 수 있다.
   1. CPU 실행: 프로세스의 잔여 burst time을 1만큼 감소시키고, 현재 CPU 총 수행량을 1만큼 증가시킨다.
   2. I/O 요청 검사: 아직 I/O를 수행하지 않은 첫 번째 CPU 수행인지와 현재 CPU 시간이 I/O 요청 시간과 동일한지, I/O 수행 시간이 0보다 큰 경우인지를 확인합니다.

   조건이 만족한다면:

   상태를 waiting(=3)으로 변경하고, I/O 완료 시점을 현재 시간 + 1+ I/O 처리 시간으로 설정한다.

잔여 CPU burst time을 계산하고 프로세스를 I/O waiting queue에 추가한다.

3. 프로세스 완료 검사: 상태를 completed(=4)로 변경한다. 완료 시간을 현재 시간 + 1로 기록한다.
4. 그 외: 위 조건에 해당하지 않으면 프로세스는 계속 실행 상태를 유지하며, 0을 반환하여 루프가 계속되도록 한다.

6. Gantt Chart 및 평가

add_gantt_entry(): 연속 실행 구간 병합한다.

Evaluation(): Gantt Chart 출력, 평균 대기 시간 및 평균 반환 시간을 계산한다.

위의 모듈들을 포함하는 시뮬레이터를 실행하면 그림 2와 같은 출력을 볼 수 있다.

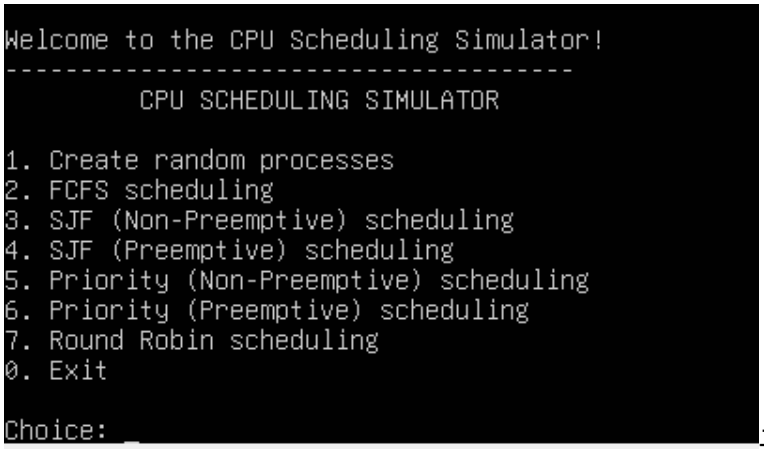

그림 2. 초기 출력 화면

가장 먼저 사용자에게 프로그램의 기능을 안내하고 선택할 수 있도록 하는 메인 메뉴 화면이 나타난다.

메뉴 옵션은 다음과 같다:

1. Create random processes: 시뮬레이션에 사용될 프로세스들을 랜덤하게 생성한다. 이 옵션을 먼저 실행해야 다른 스케줄링 알고리즘들을 테스트할 수 있다. 만약, 이 옵션을 실행하지 않고 다른 옵션을 선택한다면 1번 옵션을 먼저 선택하라는 메시지를 출력한다.

2~7. 옵션에 해당하는 알고리즘으로 스케줄링을 실행하고 스케줄링 결과가 출력된다.

0. Exit: 시뮬레이터를 종료한다.

아래에 나오는 Choice 프롬프트에 사용자가 원하는 옵션의 번호를 적으면 해당 기능이 실행된다.

예를 들어, 사용자가 먼저 1을 입력하여 프로세스를 생성한 후, 2를 입력하여 스케줄링 결과를 확인하고, 이어서 3을 입력하여 SJF 스케줄링 결과를 확인하는 방식으로 시뮬레이터를 사용할 수 있다.

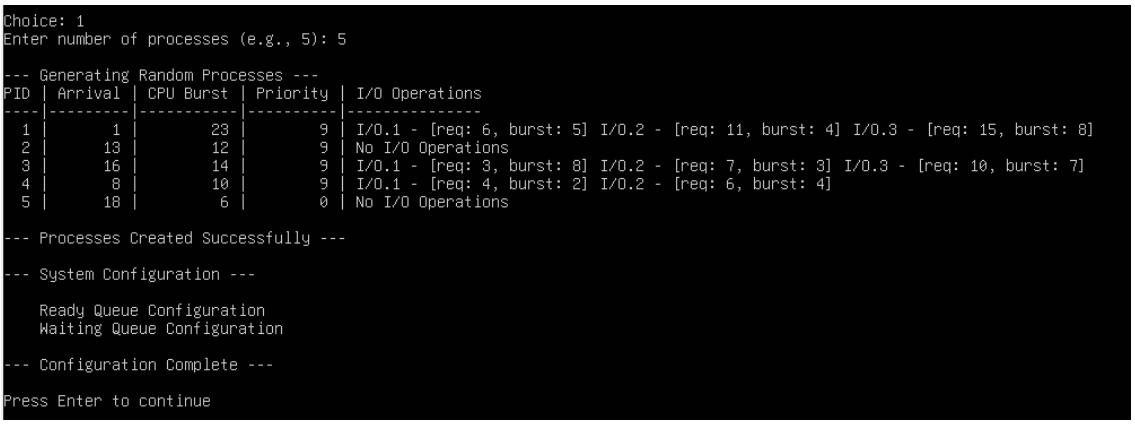

그림 3. SJF 스케줄링 결과

그림 3은 메뉴에서 1번 옵션을 실행했을 때의 출력 결과이다. 사용자가 직접 스케줄링을 실행할 프로세스의 수를 결정한다. 만약, 사용자가 프로세스 수를 결정하지 않는다면 5개가 기본값으로 결정된다. 프로세스의 개수가 결정되면 다음과 같이 생성된 프로세스 정보가 우선 출력된다.

이때, 도착 시간(Arrival), CPU Burst 시간, 우선 순위(Priority), I/O 작업 정보(I/O request 시간, I/O Burst 시간)는 모두 무작위로 생성한다. I/O 작업 발생 여부와 작업 개수 역시 무작위로 생성된다.

프로세스 정보가 출력된 이후에는 시스템 환경을 초기화하고 완료되었음을 출력한다.

Enter 키를 누르면 초기 옵션 화면이 다시 출력된다. 이후 특정 스케줄링 알고리즘 실행을 선택하면, 그림 4와 같은 결과가 출력된다.

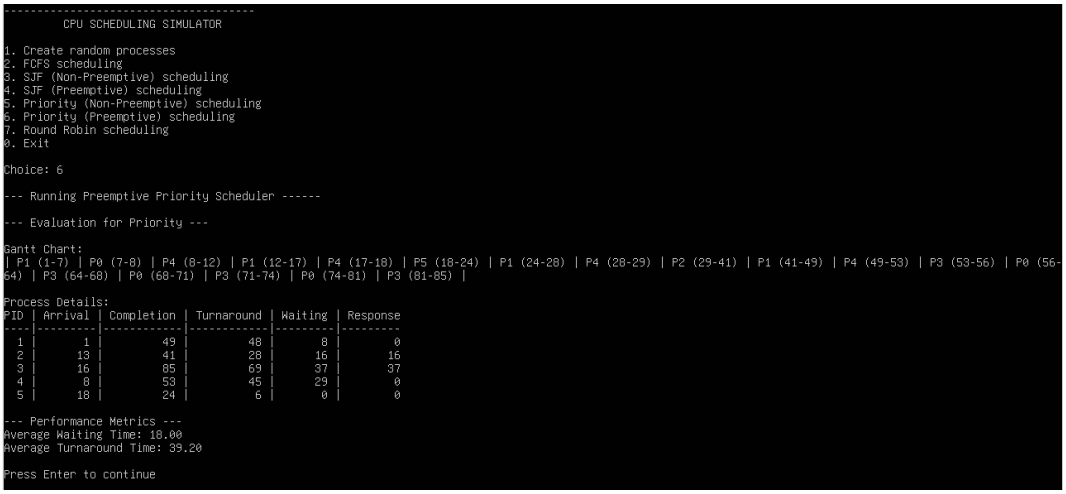

그림 4. Preemptive Priority 출력 결과

실행되는 스케줄링 알고리즘 이름, Gantt Chart, 스케줄링 결과가 표시된다. 스케줄링 결과로 각 PID의 도착 시각(Arrival time), 완료 시각(Completion time), 반환 시간 (Turnaround time), 대기 시간 (Waiting time), 응답 시간 (Response time)을 출력한다. 아래에 평균 대기 시간(Average Waiting Time), 평균 반환 시간(Average Turnaround Time)을 소수점 2자리 수까지 출력한다.

이후에 다른 알고리즘 옵션을 선택하면, 동일한 프로세스를 이용해서 스케줄링을 진행한다. 만약, 다른 프로세스로 진행하고 싶다면 옵션 1번을 선택하면 새로운 프로세스가 생성된다.

표 1은 위에서 예시로 설명한 프로세스 정보를 기준으로 작성한 알고리즘들의 동작 결과이다.

알고리즘	Average Waiting time	Average Turnaround Time
FCFS	17.80	39.00
SJF (Non-Preemptive)	15.00	36.20
SJF (Preemptive)	**14.60**	**35.80**
Priority (Non-Preemptive)	15.80	37.00
Priority (Preemptive)	18.00	39.20
Round Robin	20.20	41.40

표 1. 알고리즘 동작 결과 예시

해당 예시에서는 평균 대기 시간과 평균 반환 시간 모두 Preemptive SJF 알고리즘이 가장 우수했다.

본 시뮬레이터는 랜덤으로 프로세스를 생성하므로 실행 시마다 결과가 다를 수 있지만, 여러 번 실행하여 평균적인 경향을 관찰하면 위와 같은 이론적 특성을 확인할 수 있다. I/O 작업이 있는 프로세스의 경우, I/O 대기 시간 동안 다른 프로세스가 CPU를 사용할 수 있으므로 CPU 이용률이 향상되고 전체적인 시스템 처리량이 달라질 수 있다.

결론

본 프로젝트를 통해 FCFS, Non / Preemptive SJF, Non / Preemptive Priority, Round

Robin 총 6가지 주요 CPU 스케줄링 알고리즘을 시뮬레이션하는 프로그램을 C언어로 구현하였다. 시뮬레이터는 각 프로세스의 도착 시간(Arrival time), CPU 실행 시간 (CPU Burst time), I/O 작업 (I/O request time, I/O burst time)을 고려하여 동작한다. 각 알고리즘의 실행 과정은 간트 차트로 시각화 되고, 평균 대기 시간과 평균 반환 시간이라는 주요 성능 지표를 통해 평가된다. 일반적인 경향으로는 SJF와 Preemptive Priority가 평균 대기 시간 및 반환 시간 측면에서 우수한 성능을 보였으며, Round Robin은 응답 시간 면에서는 유리하지만 전체 반환 시간 증가를 초래함을 확인했다.

프로세스 정보는 랜덤으로 생성되어 다양한 시나리오에 대한 테스트가 가능하며, 메인 반복문을 통해 각 스케줄링 알고리즘의 핵심 기능(다음 프로세스 선택, preemptive 조건)만 분리하여 모듈성을 높였다. 특히 SJF 및 우선순위 스케줄링 알고리즘에서 다음 실행할 프로세스를 효율적으로 선택하기 위해 Ready queue와 Waiting queue를 힙(heap) 구조로 구현하였다. 이를 통해 $O(\log N)$의 시간 복잡도로 프로세스 선택 및 관리가 가능하였다. 이는 많은 수의 프로세스가 존재할 경우 시뮬레이션의 성능 및 실제 스케줄러의 효율성을 크게 향상시킨다. 이를 통해 각 알고리즘의 특징과 장단점을 실제 수치를 통해 비교하고 이해하는 데 도움을 줄 수 있었다.

CPU 스케줄링은 운영체제의 핵심 개념 중 하나로, 이번 프로젝트를 통해 이론으로만 학습했던 다양한 알고리즘들의 실제 동작 방식을 코드로 구현해보면서 각 알고리즘의 특성과 복잡성을 더 깊이 이해할 수 있었다. 특히 Preemptive 알고리즘의 구현, 프로세스 상태 관리 (ready, execute, waiting, complete), I/O 처리 등은 구현에 어려움을 겪었기에 많은 신경을 써야 했다. 여러 알고리즘을 하나의 프레임워크에서 실행하고 비교할 수 있도록 설계하는 과정에서 소프트웨어 공학적인 고민도 해볼 수 있는 좋은 기회였다.

본 시뮬레이터를 발전시키기 위해 다음과 같은 추가적인 기능을 구현해볼 수 있을 것이다.

1. 에이징(Aging): SJF나 Priority 스케줄링에서 발생할 수 있는 기아(starvation) 현상을 완화하기 위해 에이징 기법을 추가할 수 있다.
2. EDF (Earliest Deadline First): 실시간 시스템에서 중요한 알고리즘으로, 각 프로세스의 Deadline을 기준으로 가장 마감 시간이 임박한 프로세스에게 CPU를 할당하는 Preemptive scheduling이다. 이를 구현하여 실시간 스케줄링을 시뮬레이션해 볼 수 있다.
3. Lottery Scheduling: 각 프로세스에게 우선순위나 자원 요구량에 비례하여 '복권(lottery ticket)'을 할당하고, 무작위로 복권을 추첨하여 당첨된 프로세스가 자원을 사용하는 확률적 스케줄링 기법이다. 확률적 요소와 자원 분배 메커니즘 구현을 경험해 볼 수 있다.

이러한 개선 사항들을 통해 다양한 기능을 지원하는 시뮬레이터로 발전시킬 수 있을 것이다.

참고문헌

yousefkotp / CPU-Scheduling-Algorithms . (n.d.). https://github.com/yousefkotp/CPU-Scheduling-Algorithms.

A.Silberschatz, P.Galvin, & G.Gagne. (2019). *Operating System Concepts (10th Ed)* (pp. 200-214, 227-234, P-30-P-31). n.p.: Wiley.

고정국. (2012). CPUSim: CPU 스케줄링 알고리즘 교육을 지원하는 시뮬레이터. 한국정보통신학회논문지, 16(4), 835-842.