[2018-2] OS Assignment #2



**운영체제 (공통)**

#1 운영체제 스케줄러 알고리즘을 시뮬레이션 하는 프로그램을 구현하시오.

(PR 스케줄링 알고리즘 추가 구현)

|  |  |
| --- | --- |
| 교과목명 | 운영체제(공통) |
| 담당교수 | 양 승 민 교수 |
| 학 과 | 컴 퓨 터 학부 |
| 학 번 | 2 0 1 6 3 3 4 1 |
| 이 름 | 강 지 현 |
| 제출일자 | 2018. 10. 19 |



목차

소개 3

관련 연구

가) 소스분석 4

나) 스케줄링 알고리즘 설명 12

다) API 함수 설명 14

추가 기능 구현 20

실행화면 21

소개

본 프로그램은 리눅스 운영체제에서 동일 폴더 내의 txt파일을 한 줄씩 읽으면서 프로세스의 id와 해당 프로세스의 도착시간(arrive\_time), 서비스시간(service\_time), 우선순위(priority) 등의 여러 항목을 갖고 프로세스 스케줄링 알고리즘에 따라 실행할 경우 프로세스별로 실행되는 순서와 총 소요시간, 평균 소요시간을 출력해주는 프로세스 스케쥴링 시뮬레이션 프로그램이다. 스케쥴링 알고리즘은 SJF, SRT, RR, PR총 네 가지로 SJF를 제외한 나머지는 모두 선점(preemptive) 방식이다.

추가로 구현한 PR (priority), 즉 우선순위 알고리즘은 선점 방식이므로, 도착 시간에 따라 큐 (queue)에서 추가되는 프로세스가 우선순위가 높으면 해당 프로세스로 큐 순서를 변경하고, 우선순위가 같은 경우는 서비스시간이 짧은 프로세스를 먼저 실행하는, 즉 SJF를 혼합한 방식을 택하였다.

네 가지의 스케쥴링 알고리즘의 구동방식과 성능확인을 위해 간트(Gantt) 차트형식으로 출력하도록 하였는데, 프로세스별로 리소스를 할당받는 것을 나타낼 때 '\*' 출력이 되도록 하였다. 또한 프로세스별로 평균 소요시간(avg\_turnaround\_time)과 평균 대기시간(avg\_waiting\_time)도 출력하여 각 알고리즘 별로 성능향상이 있는지 한 눈에 확인이 가능하도록 하였다.

관련 연구

* 1. 소스 분석
     1. sched.c
* **enum** 
  + - 1. SCHED\_SJF는 0, SCHED\_SRT는 1, SCHED\_RR 는 2, SCHED\_PR는 3, SCHED\_MAX는 4
* **구조체 \_Process**  
  구조체 Process로 재정의하여 사용  
  프로세스의 정보를 멤버로 갖는 구조체로 크게 세 가지 역할로 나눌 수 있다.   
  하기의 표에서 각 역할별 변수들의 정보를 확인할 수 있다.

|  |  |  |  |
| --- | --- | --- | --- |
| * + - 1. 역할 | * + - 1. 데이터 형 (data type) | * + - 1. 멤버 변수 명 (variable name) | * + - 1. 정보 (information) |
| * + - 1. 프로세스       2. 개수 확인용 | * + - 1. int | * + - 1. idx | * + - 1. 읽은 순서대로 번호 부여받음       2. 0부터 반영되며, 프로세스가 추가 될 때마다 1씩 증가 한 값을 부여받음 |
| * + - 1. int | * + - 1. queue\_idx | * + - 1. 큐에 들어가는 번호로, 스케쥴링 알고리즘에 따라 변경 될 수 있음. |
| * + - 1. 프로세스       2. 자체의       3. 상세정보 | * + - 1. char | * + - 1. id[ID\_MAX+1] | 프로세스의 id를 나타내는 것으로, 널문자('\0') 포함 3개의 char 원소 가짐 |
| * + - 1. int | * + - 1. arrive\_time | * + - 1. 프로세스가 cpu\_time 기준으로       2. 큐에 도착하는 시간 |
| * + - 1. int | * + - 1. service\_time | * + - 1. 프로세스가 cpu\_time 기준으로       2. 실행되는데 필요한 시간 |
| * + - 1. int | * + - 1. priority | * + - 1. 프로세스의 우선순위 |
| * + - 1. 프로세스 실행과 종료사이       2. 정보 저장용 | * + - 1. int | * + - 1. remain\_time | * + - 1. 프로세스가 종료될 때까지 남은 시간 |
| * + - 1. int | * + - 1. complete\_time | * + - 1. 프로세스가 cpu\_time을 기준으로       2. 작업을 모두 끝낸 시각 |
| * + - 1. int | * + - 1. turnaround\_time | * + - 1. 프로세스가 완료될 때까지 쓴 총 시간으로, 실제 실행 시간+대기 시간이 적용된 소요시간 |
| * + - 1. int | * + - 1. wait\_time | * + - 1. 도착한 프로세스가 할당 조건에 밀려 preemption 당해 큐로 돌아갔다가 다시 실행될 때까지 걸리는 시간. 즉, 대기시간을 뜻함 |

* **1차원 구조체 배열 processes[PROCESS\_MAX]**  
  static 키워드를 사용하여 선언과 함께 생성 후 프로그램이 종료될 때까지 유지되며,  
  본 프로그램에서는 프로세스가 추가될 때에 순서대로 담아놀 자료구조이다.
* **int형 변수 process\_total**  
  static 키워드를 사용하여 선언과 함께 생성 후 프로그램이 종료될 때까지 유지되며,  
  전역에서 선언하였음으로 기본값은 0이다. read\_config 함수를 실행할 때마다 0으로 초기화되며, append\_process 함수에서 프로세스가 추가될 때마다 후위증가연산자로 1씩 증가한다.
* **1차원 구조체 포인터 배열 queue[PROCESS\_MAX]**  
  static 키워드를 사용하여 선언과 함께 생성 후 프로그램이 종료될 때까지 유지되며,  
  전역에서 선언하였음으로 기본값은 0으로 NULL, 즉 아무것도 가리키지 않는 셈이 된다. simulate 함수에서 모든 원소가 NULL로 초기화 되며, cpu\_time에 따라 프로세스가 도착하는대로 queue 배열의 원소가 해당 프로세스를 가리킬 수 있도록 추가된다.
* **int형 변수 queue\_len**  
  static 키워드를 사용하여 선언과 함께 생성 후 프로그램이 종료될 때까지 유지되며,  
  전역에서 선언하였음으로 기본값은 0이다. simulate 함수 윗부분에서 0으로 초기화 되며, 프로세스가 더해질 때마다 후위증가연산자로 1씩 증가한다. 또한 프로세스가 스케쥴링 알고리즘에서 실행되어 종료 될 때에는 큐 길이가 줄어들기 때문에 프로세스에서 실행할 때 필요한 시간이 끝나면 후위감소연산자로 1씩 감소한다.
* **2차원 char 배열 schedule[PROCESS\_MAX][SLOT\_MAX]**  
  static 키워드를 사용하여 선언과 함께 생성 후 프로그램이 종료될 때까지 유지되며,  
  전역에서 선언하였음으로 기본값은 모든 원소가 0이다. simulate 함수 윗부분에서 모든 원소가 0으로 초기화 된다. 프로세스 별로 실행되는 순서를 시각화 하기 위해 최대 대기시간 등을 고려하여 최대 갯수인 프로세스 별로 (30+30)\*260개의 칸을 가진다. 이후 간트 차트로 스케쥴링 알고리즘의 성능을 나타낼 때에 '\*' 문자를 넣어 cpu\_time 기준으로 실행되는 프로세스를 나타내주는 자료구조이다.
* **함수 strstrip**

|  |  |
| --- | --- |
| * + - * 1. 헤더 | * + - * 1. 사용자 정의 함수 |
| * + - * 1. 형태 | * + - * 1. static char\* strstrip(char\* str) |
| * + - * 1. 인수 | * + - * 1. char\* str |
| * + - * 1. 반환 | * + - * 1. char\* |
| * + - * 1. 설명 | * + - * 1. 인수로 받은 str을 가져와 문장끝('\0')까지 읽고 반환. |

* **함수 check\_valid\_id**

|  |  |
| --- | --- |
| * + - * 1. 헤더 | * + - * 1. 사용자 정의 함수 |
| * + - * 1. 형태 | * + - * 1. static int check\_valid\_id (const char \*str) |
| * + - * 1. 인수 | * + - * 1. const char\* str |
| * + - * 1. 반환 | * + - * 1. int |
| * + - * 1. 설명 | * + - * 1. 인수로 받은 str이 지정된 id 문자열의 문자 갯수가 2~8사이에 해당하는지, 소문자와 숫자만 사용하였는지 확인하여 소문자 또는 숫자만 있는 경우 0, 이외의 문자가 있는 경우 -1을 반환. |

* **함수 lookup\_process**

|  |  |
| --- | --- |
| * + - * 1. 헤더 | * + - * 1. 사용자 정의 함수 |
| * + - * 1. 형태 | * + - * 1. static Process\* lookup\_process (const char \*id) |
| * + - * 1. 인수 | * + - * 1. const char\* id |
| * + - * 1. 반환 | * + - * 1. Process \* |
| * + - * 1. 설명 | * + - * 1. 인수로 받은 const char\* id가 지금까지 추가된 processes의 원소들인 Process들, 즉 프로세스의 id와 같은 것이 있는지 확인하여 일치하는 것이 있는 경우, 해당 processes의 원소 주소를 반환하고 일치하는 것이 없는 경우는 NULL을 반환. |

* **함수 append\_process**

|  |  |
| --- | --- |
| * + - * 1. 헤더 | * + - * 1. 사용자 정의 함수 |
| * + - * 1. 형태 | * + - * 1. static void append\_process (Process\* process) |
| * + - * 1. 인수 | * + - * 1. Process\* process |
| * + - * 1. 반환 | * + - * 1. void (없음) |
| * + - * 1. 설명 | 인수로 받은 process를 processes 배열에 추가. 실행순서는 다음과 같다.   1. processes배열의 process\_total번째 원소에 인수의 멤버 변수 값 복사 2. processes 배열의 process\_tota번째 원소의 멤버 idx에 process\_total 값 복사 3. 후위증가연산자로 process\_total 값 1 증가 (static 변수로 값 계속 반영) |

* **함수 read\_config**

|  |  |
| --- | --- |
| * + - * 1. 헤더 | 사용자 정의 함수 |
| * + - * 1. 형태 | static int read\_config (const char \*filename) |
| * + - * 1. 인수 | pid\_t pid |
| * + - * 1. 반환 | int |
| * + - * 1. 설명 | 인수로 받은 filename으로 된 파일을 열어서 한 줄씩 읽으며 문장 양식에 맞춰 프로세스 추가 작업을 수행한다. id, arrive\_time, service\_time, priority 항목을 모두 확인 하고 유효한 값인 경우process의 멤버에 대입한다.  함수의 실행순서는 크게 다음과 같다.   1. fopen함수에 'r'(read) 모드, 즉 읽기 전용으로 파일 읽기 (파일 열기 실패한 경우 -1 반환) 2. while문에 fgets 함수 넣어 아래의 경우가 해당될 때까지 한줄씩 읽고, while문 안 문장 실행  - '\n ' 만나거나 - line 길이 다 끝날 때까지 3. '#' 문자나 빈 문장이면 **continue** 로 while의 조건문으로 jump 4. id확인 및 프로세스 id에 값 복사 - strstrip 함수 사용해 문장 끝('\0')까지 읽고 반환 - strchr함수로 ' ' 없으면 -1반환, goto문을 사용해 **invalid\_line**으로. - check\_valid\_id 함수 사용해 유효하지 않은 문자일 경우 MSG함수 사용해 오류 메시지 띄우고 **continue** 로 while의 조건문으로 jump - lookup\_process 함수 사용해 중복된 이름의 id일 경우 MSG함수 사용해 오류 메시지 띄우고**continue** 로 while의 조건문으로 jump - strcpy함수로 해당 process 의 멤버 id에 strstrip으로 추출된 문자열 복사. 5. arrive time 확인 및 프로세스 arrive\_time에 값 복사 - strstrip 함수 사용해 문장 끝('\0')까지 읽고 반환 - strchr함수로 ' ' 없으면 -1반환, goto문을 사용해 **invalid\_line**으로. - strtol 함수로 strstrip으로 추출된 문자열을 long형, 10진수로 반환하고, 프로세스 arrive\_time에 값을 복사한다. 단, 복사된 값이 ARRIVE\_TIME\_MAX보다 크거나 ARRIVE\_TIME\_MIN보다 작거나, processes 배열에 추가되기 바로 전 원소 즉, 직전의 프로세스보다 작은경우(도착시간 오류)는 MSG 함수 사용해 오류 메시지 띄우고 **continue** 로 while의 조건문으로 jump 6. service time 확인 및 프로세스 service\_time에 값 복사 - strstrip 함수 사용해 문장 끝('\0')까지 읽고 반환 - strchr함수로 ' ' 없으면 -1반환되고 goto문을 사용해 **invalid\_line**으로. - strtol 함수로 strstrip으로 추출된 문자열을 long형, 10진수로 반환하고, 프로세스service\_time에 값을 복사한다. 단, 복사된 값이 SERVICE\_TIME\_MAX보다 크거나 SERVICE\_TIME\_MIN보다 작으면, MSG 함수 사용해 오류 메시지 띄우고 **continue** 로 while의 조건문으로 jump 7. priority 확인 및 프로세스 priority에 값 복사 - strstrip 함수 사용해 문장 끝('\0')까지 읽고 반환 - strtol 함수로 strstrip으로 추출된 문자열을 long형, 10진수로 반환하고, 프로세스service\_time에 값을 복사한다. 단, 복사된 값이 PRIORITY\_TIME\_MIN보다 작거나, SERVICE\_TIME\_MAX보다 크면, MSG 함수 사용해 오류 메시지 띄우고 **continue** 로 while의 조건문으로 jump 8. append\_process 함수로 현재 process 추가 및 **continue** 9. while문이 끝나면 fclose 함수로 열어둔 파일 닫기 |

* **함수 simulate**

|  |  |
| --- | --- |
| * + - * 1. 헤더 | * + - * 1. 사용자 정의 함수 |
| * + - * 1. 형태 | * + - * 1. static void simulate (int sched) |
| * + - * 1. 인수 | * + - * 1. int sched |
| * + - * 1. 반환 | * + - * 1. void (없음) |
| * + - * 1. 설명 | 인수로 받은 sched 값에 따라 switch문으로 프로세스 스케쥴링 알고리즘을 시뮬레이션 하는 함수이다. 함수의 실행순서는 크게 다음과 같다.   1. schedule 배열 및 지역변수 초기화 2. 프로세스가 완료된 값(p\_done)이 지금까지 추가된 프로세스(process\_total)의 값이 작을 때까지 아래의 문장 반복  * Process\* 형 변수 pp선언 * pp가 processes의 p번째 원소가 가리키는 것과 같은 Process 가리키도록 주소 대입 * cpu\_time pp의 도착시간이 같지 않으면 반복문 탈출 * pp가 가리키는 Process의 remain\_time에 pp의 service\_time 대입 * queue[queue\_len] 원소가 pp와 같은 Process 가리키게 함 * 후위증가연산자로 queue\_len 1씩 증가  1. switch문에 sched에 해당하는 값의 case문 실행  * SCHED\_SJF 일 경우 : process가 NULL이 아닐 때까지, 현재 shortest 값보다 queue[i]의 멤버 service\_time이 짧은 경우 process가 queue[i]가 가리키는 Process를 같이 가리키도록 주소값 복사. shortest 값은 현재 process의 service\_time 대입 * SCHED\_SRT 일 경우 : queue\_len이 후위증가연산으로 커지는 I 값보다 작을 때까지 queue[i]원소의 remain\_time이 shortest보다 작은경우, process가 queue[i]가 가리키는 Process를 같이 가리키도록 주소값 복사. shortest 값은 현재 process의 remain\_time 대입 * SCHED\_RR 일 경우 : process를 우선 queue의 0번째 원소가 가리키는 Process를 같이 가리키도록 주소값 복사 i 값이 queue\_len-1 보다 작을 때까지 아래의 문장 반복 - queue[i] 원소가 queue[i+1] 원소가 가리키는 Process를 가리키게 함   - queue[i] 원소의 queue\_idx에 현재 i 값 대입  현재 process가 가리키는 Process를 queue[i]도 가리키게 함  queue[i] 원 원소의 queue\_idx에 현재 i 값 대입   * SCHED\_PR은 추가기능 구현에 기재함. * default 값, 즉 위의 4개의 값 이외의 값 입력할 경우 MSG로 오류메시지 출력  1. 스케쥴링할 프로세스가 없을 경우 continue로 cpu\_time 증가 후 다시 반복문 실행 2. schedule 배열의 현재 process의 idx번째 값을 행으로, cpu\_time을 열로 가지는 원소에 1 대입 3. 현재 process의 remain\_time을 후위감소연산자로 1씩 감소시킴 4. 현재 process의 remain\_time이 0보다 작거나 같은 경우, 현재 process가 완료된 것이므로, 다음 queue를 실행시키도록 한다. 이 때, 대기열은 줄어든 것이므로 queue\_len은 후위감소연산자로 1씩 감소시킨다. 5. 현재 process의 complete\_time 즉, 완료시각은 cpu\_time에 1을 더한 값을 대입 6. 현재 process의 turnaround\_time 즉, 소요시간은 현재 process의 complete\_time에서 현재 process의 arrive\_time을 뺀 값을 대입. 7. 현재 process의 wait\_time은 현재 process의 소요시간(turnaround\_time) – 현재 process의 도착시각(arrive\_time)한 값을 대입 8. 현재 process를 NULL, 즉 아무것도 가리키지 않도록 설정하고, 완료된 프로세스에 대한 처리가 끝났으므로 후위증가연산자로 p\_done 값을 1씩 증가시키도록 한다. 9. 삼항 연산자로 sched이 enum에 명시된 값과 같은 경우 각 스케쥴링 알고리즘 문자열을 출력하게 한다. 10. sum\_turnaround\_time, sum\_waiting\_time 값 초기화 11. 간트 차트(Gantt chart)출력 : 이 때도 삼항 연산자로 schedule 배열의 p행 slot이 1이면 '\*', 0이면 ' '문자 출력 12. sum\_turunaround\_time에 processes 배열의 p번째 원소의 turnaround\_time 값 더해감 (processes배열에 추가된 Process의 turnaround\_time 모두 더함) 13. sum\_waiting\_time 에 processes 배열의 p번째 원소의 waiting\_time 값 더해감 (processes배열에 추가된 Process의 waiting\_time 모두 더함) 14. avg\_turnaround\_time으로 소요시간 합 / 추가된 프로세스 갯수 로 나누고 float로 캐스팅된 값 대입 15. avg\_waiting\_time으로 대기시간 합 / 추가된 프로세스 갯수로 나누고 float로 캐스팅된 값 대입 16. printf 함수로cpu\_time, avg\_turnaround\_time, avg\_waiting\_time 값 출력 하여 성능 비교할 수 있도록 함 |

* **함수 main**

|  |  |
| --- | --- |
| 헤더 | 사용자 정의 함수 |
| 형태 | int main (int argc, char\*\* argv) |
| 인수 | int argc, char\*\* argv |
| 반환 | void (없음) |
| 설명 | 인수로 받은 문자열 배열에 있는 문장을 하나씩 읽는다.  terminal에서 프로그램 실행과 읽어올 txt파일을 띄어쓰기로 구분하여 읽게 하였는데, 프로그램 실행 명령어만 적을 경우는 오류메시지 띄우고 -1을 반환하여 비정상 종료되도록 하였다.  이 때, read\_config함수를 이용하였는데, 읽기 실패한 경우에만 MSG로 오류메시지를 띄우고 -1을 반환하여 비정상 종료 . sched가 SCHED\_MAX의 값보다 작을 때까지 simulate 함수 실행 하며 simulate의 실인자는 sched가 되도록 하였다. |

* 1. **스케줄링 알고리즘 설명  
     [예시 프로세스 목록]**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **process id** | **arrive time** | **service time** | **priority** | **color** |
| **P1** | **0** | **3** | **1** |  |
| **P2** | **2** | **6** | **2** |  |
| **P3** | **4** | **4** | **1** |  |
| **P4** | **6** | **5** | **3** |  |
| **P5** | **8** | **2** | **1** |  |

* + 1. **SJF (Shortest Job First) scheduling algorithm**: 짧은 서비스시간을 가지고 있는 프로세스가 먼저 실행되는 것으로,  
       비선점(non-preemptive) 방식이다. 상기의 프로세스 목록을 실행 시 결과는 다음과 같다.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| cpu\_time | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| process |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| arrive | P1 |  | P2 |  | P3 |  | p4 |  | p5 |  |  |  |  |  |  |  |  |  |  |  |  |

* + 1. **SRT (Shortest Remaining Time first) scheduling algorithm**: 남은시간이 가장 짧은 프로세스가 먼저 실행되는 것으로, 선점(preemptive) 방식이다. 이 때 주의해야 할 점은 프로세스가 도착시간에 바로 서비스 시간을 알 수 있는 것이 아니라, +1의 시간이 지나야 새로 들어온 프로세스 서비스 시간을 알 수 있다는 점이다. 상기의 프로세스 목록을 실행 시 결과는 다음과 같다.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| cpu\_time | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| process |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| arrive | P1 |  | P2 |  | P3 |  | p4 |  | p5 |  |  |  |  |  |  |  |  |  |  |  |  |

* + 1. **RR (Round Robin) scheduling algorithm**: 시간 단위를 나누어 시간 단위만큼 공평하게 프로세스를 실행해주는 것으로 일단 프로세스가 도착하면, 그 이후에는 공평하게 시간 단위 별로 프로세스가 번갈아 실행되게 되므로, 선점(preemptive)방식이다. 이 때 주의해야 할 점은 프로세스가 도착한 뒤, 바로 실행되는 것이 아니라, 그전에 실행되는 프로세스에게 할당된 시간이 끝나면서 바로 실행되는 것이므로, 단위시간만큼의 딜레이가 발생한다.  
       상기의 프로세스 목록을 실행 시 결과는 다음과 같다.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| cpu\_time | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| process |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| arrive | P1 |  | P2 |  | P3 |  | p4 |  | p5 |  |  |  |  |  |  |  |  |  |  |  |  |

* + 1. **PR (Priority) scheduling algorithm**: 프로세스의 우선순위가 높은 것이 먼저 실행되는 방식으로, 프로세스가 도착하면 프로세스의 우선순위를 비교해서 우선순위가 높은 것이 먼저 실행되게 된다. 이 때 기본적으로는 비선점(non-preemptive) 방식이나, 본 프로그램에서는 선점(preemptive)방식을 사용하였다. 상기의 프로세스목록을 비선점, 선점방식으로 실행 시 결과는 다음과 같다.
* **선점(preemptive)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| cpu\_time | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| process |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| arrive | P1 |  | P2 |  | P3 |  | p4 |  | p5 |  |  |  |  |  |  |  |  |  |  |  |  |

* **비선점(non-preemtive)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| cpu\_time | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| process |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| arrive | P1 |  | P2 |  | P3 |  | p4 |  | p5 |  |  |  |  |  |  |  |  |  |  |  |  |

* 1. **API 함수 설명**
     1. **strlen 함수**

|  |  |
| --- | --- |
| * + - * 1. 헤더 | * + - * 1. string.h |
| * + - * 1. 형태 | * + - * 1. size\_t strlen(const char \*s) |
| * + - * 1. 인수 | * + - * 1. const char \*s |
| * + - * 1. 반환 | * + - * 1. size\_t (문자열의 길이) |
| * + - * 1. 설명 | 문자열 s의 길이를 계산한다. 이 때, NULL문자의 길이는 포함되지 않는다. |

* + 1. **strchr 함수**

|  |  |
| --- | --- |
| * + - * 1. 헤더 | * + - * 1. string.h |
| * + - * 1. 형태 | * + - * 1. char\* strchr(const char\* s, int c) |
| * + - * 1. 인수 | * + - * 1. const char \*s |
| * + - * 1. 반환 | * + - * 1. char\*(문자열 s에서 발견된 문자 c의 포인터) |
| * + - * 1. 설명 | 문자 배열 s내에 문자 c가 있는지 검사하고 있을 경우 문자 c가 있는 번지를 반환한다. 만약 문자열 s에 문자 c가 발견되지 않을 경우 NULL을 리턴한다. |

* + 1. **strcmp 함수**

|  |  |
| --- | --- |
| * + - * 1. 헤더 | * + - * 1. string.h |
| * + - * 1. 형태 | * + - * 1. int strcmp(const char\* s1, const char\* s2) |
| * + - * 1. 인수 | * + - * 1. const char\* s1, const char\* s2 |
| * + - * 1. 반환 | * + - * 1. s1<s2 인 경우 음수 반환         2. s1=s2 인 경우 0을 반환         3. s1>s2인 경우 양수 반환 |
| * + - * 1. 설명 | 인자로 받은 두개의 문자열 s1, s2를 대소 비교한다. 대문자,소문자 구분하며, 대소비교보다는 함수결과에 ! (NOT)연산자를 붙여 문자열의 일치여부를 확인하는데 사용한다. |

* + 1. **strcpy 함수**

|  |  |
| --- | --- |
| * + - * 1. 헤더 | * + - * 1. string.h |
| * + - * 1. 형태 | * + - * 1. char \*strcpy(char \*dest, const char\* src); |
| * + - * 1. 인수 | * + - * 1. char \*dest, const char\* src |
| * + - * 1. 반환 | * + - * 1. char\* (dest+strlen(src)의 번지 반환) |
| * + - * 1. 설명 | 문자열 src를 문자배열 dest로 복사한다. NULL문자도 함께 복사가 된다. dest 크기가 src 크기보다 같거나 커야 복사가 온전히 일어나며, 만일 dest 문자열 크기가 src문자열 크기보다 작은 경우 dest 뒤의 인접데이터가 손실된다. |

* + 1. **strtol 함수**

|  |  |
| --- | --- |
| * + - * 1. 헤더 | * + - * 1. stdlib.h |
| * + - * 1. 형태 | * + - * 1. long strtol (const char\*s, char\*\* endptr, int radix) |
| * + - * 1. 인수 | * + - * 1. const char\*s, char\*\* endptr, int radix |
| * + - * 1. 반환 | * + - * 1. long (변환된 long형 정수 반환) |
| * + - * 1. 설명 | 인자로 받은 문자열 s을 radix로 받은 진법으로 전환하여 long형 값으로 반환한다. 이 때 불필요한 문자가 문자열사이에 있거나, 문자열이 [공백,tab  ][부호][숫자] 순서로 나와있지 않은 경우는 변환이 중지되고, 중지된 위치를 endptr에 설정해준다. 따라서 endptr은 디버깅시 이용할 수 있다. 본 프로그램에서는 10진수로 사용하였는데, radix가 0인 경우는 숫자 형태에 따라 진법을 구분한다. 0abc인 경우는 8진수로, 0xabc일 때는 16진수로, abcdlf EOsms 10진수로 인식하게 된다. |

* + 1. **isspace 함수**

|  |  |
| --- | --- |
| * + - * 1. 헤더 | * + - * 1. ctype.h |
| * + - * 1. 형태 | * + - * 1. int isspace(int c) |
| * + - * 1. 인수 | * + - * 1. int c |
| * + - * 1. 반환 | * + - * 1. int (c가 공백문자이면 0이외의 값(참) 리턴, 아니면 0(거짓) 반환) |
| * + - * 1. 설명 | 문자 c가 공백 문자인지를 검사해준다. 공백문자는 space, TAB, CR, LF, VT(수직탭)으로, 이 문자들이 나오면 0이외의 값(참)을, 그 이외의 문자가 나오면 0(거짓)을 반환한다. |

* + 1. **isupper 함수**

|  |  |
| --- | --- |
| * + - * 1. 헤더 | * + - * 1. ctype.h |
| * + - * 1. 형태 | * + - * 1. int isupper(int c) |
| * + - * 1. 인수 | * + - * 1. int c |
| * + - * 1. 반환 | * + - * 1. int (c가 A~Z이면 0이외의 값(참) 리턴, 아니면 0(거짓) 반환) |
| * + - * 1. 설명 | 문자 c가 대문자인지를 검사해준다. 그 외의 문자일 경우는 0을 반환한다 |

* + 1. **isdigit 함수**

|  |  |
| --- | --- |
| * + - * 1. 헤더 | * + - * 1. ctype.h |
| * + - * 1. 형태 | * + - * 1. int isdigit(int c) |
| * + - * 1. 인수 | * + - * 1. int c |
| * + - * 1. 반환 | * + - * 1. int (c가0~9 사이의 문자면 0이외의 값(참) 리턴, 그 외의 문자면0(거짓) 반환) |
| * + - * 1. 설명 | 문자 c가 숫자인지를 검사해준다. 그 외의 문자일 경우는 0을 반환한다 |

* + 1. **memmove 함수**

|  |  |
| --- | --- |
| * + - * 1. 헤더 | * + - * 1. string.h |
| * + - * 1. 형태 | void\* memmove (void \*dest, const void\*src,, size\_t n); |
| * + - * 1. 인수 | * + - * 1. int c |
| * + - * 1. 반환 | * + - * 1. void\* (dest 번지를 반환하며, 실패하면 NULL 포인터 반환) |
| * + - * 1. 설명 | src번지에 있는 데이터를 dest가 지정하는 번지로 n 바이트만큼 복사  str번지에 있는 데이터를 start가 지정하는 번지로,  src+(src의 데이터형인char형(1byte) \*1)바이트만큼 복사 |

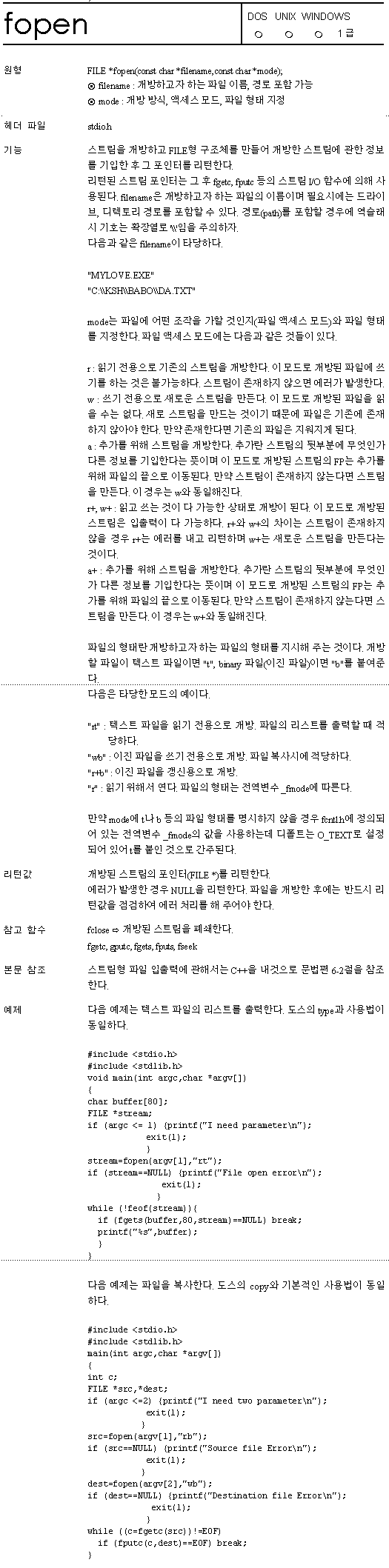
* + 1. **memset 함수**

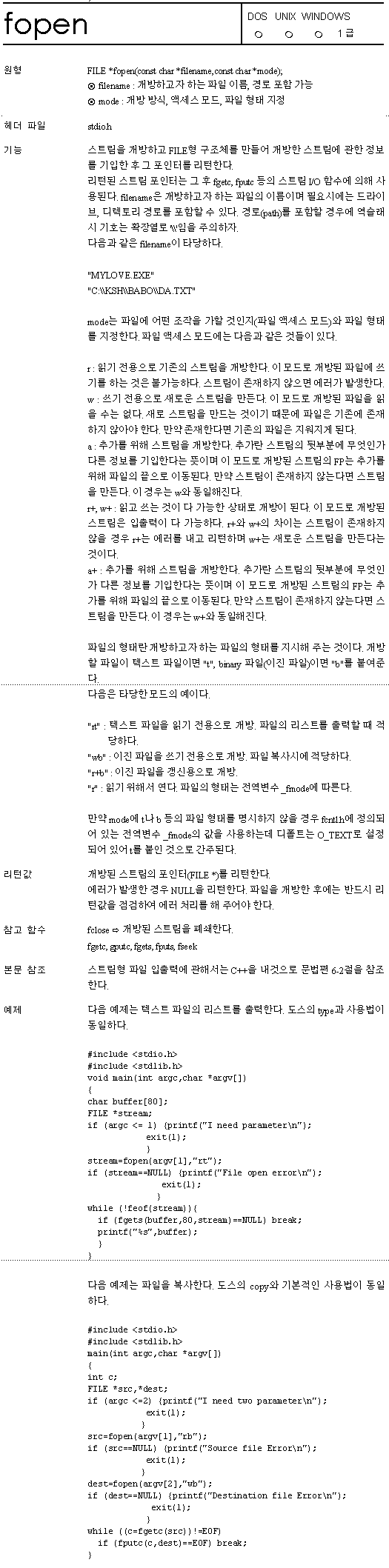
|  |  |
| --- | --- |
| * + - * 1. 헤더 | * + - * 1. string.h |
| * + - * 1. 형태 | void\* memset (void \*dest, int c, size\_t n); |
| * + - * 1. 인수 | * + - * 1. void \*dest, int c, size\_t n |
| * + - * 1. 반환 | * + - * 1. void\* dest (dest 번지) |
| * + - * 1. 설명 | 보통 할당받은 메모리를 특정 값(c)으로 초기화 하는 것으로, NULL로 초기화 할 때에 많이 사용. 디버깅에 유리함. dest가 올바르지 않은 경우 NULL 포인터 반환 |

* + 1. **fgets 함수**

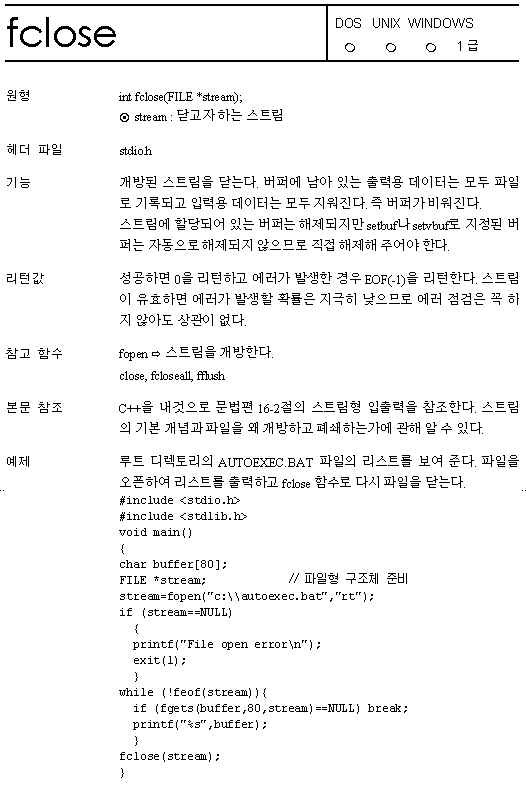
|  |  |
| --- | --- |
| * + - * 1. 헤더 | * + - * 1. stdio.h |
| * + - * 1. 형태 | * + - * 1. char \*fgets(char \*s, int n FILE\* stream) |
| * + - * 1. 인수 | * + - * 1. char \*s, int n FILE\* stream |
| * + - * 1. 반환 | * + - * 1. char\* (배열 s의 문자열을 반환) |
| * + - * 1. 설명 | stream에서 문자열을 읽어와 s에 기록한다. 이 때, '\n'과 같은 개행문자를 만날 때까지 문자열을 읽어들이거나, '\0' 과같이 문자열의 끝까지 읽어들이는데, 읽어들인 문자열 맨끝에는 NULL이 삽입된다. 읽어들이는 문자의 최대 갯수는 n-1개 이다. |

* + 1. **fopen 함수**

****

****

* + 1. **fclose 함수**

****

**3. 추가 기능 구현 방법**

PR, 즉 우선순위 스케쥴링 알고리즘의 구동원리는 다음과 같다.

1) priority 정렬을 위한 임시변수 설정

: 블럭 내의 int형 변수 tmp\_PR을 설정하고 PRIORITY\_MAX+1, 즉 priority 최대값(최하위) + 1 값 주었음

1. priority가 같을 경우의 임의 규칙 설정  
   (service\_time이 짧은 것을 먼저 실행하는 SJF scheduling algorithm 적용)

:블럭 내의 int형 변수 tmp\_service을 설정하고 SERVICE\_TIME\_MAX+1, 즉 service\_time 최대값(30) + 1 값 주었음

1. queue 를 순회

: queue[i]가 가리키는 Process 구조체의 멤버 priority와 tmp\_PR을 비교

\* priority가 tmp\_PR보다 작은 경우

- 현재 process가 queue[i]가 가리키는 것과 같은 Process 구조체 가리키도록 함

- tmp\_PR에 process가 가리키는 Process 구조체의 멤버 priority 값 복사

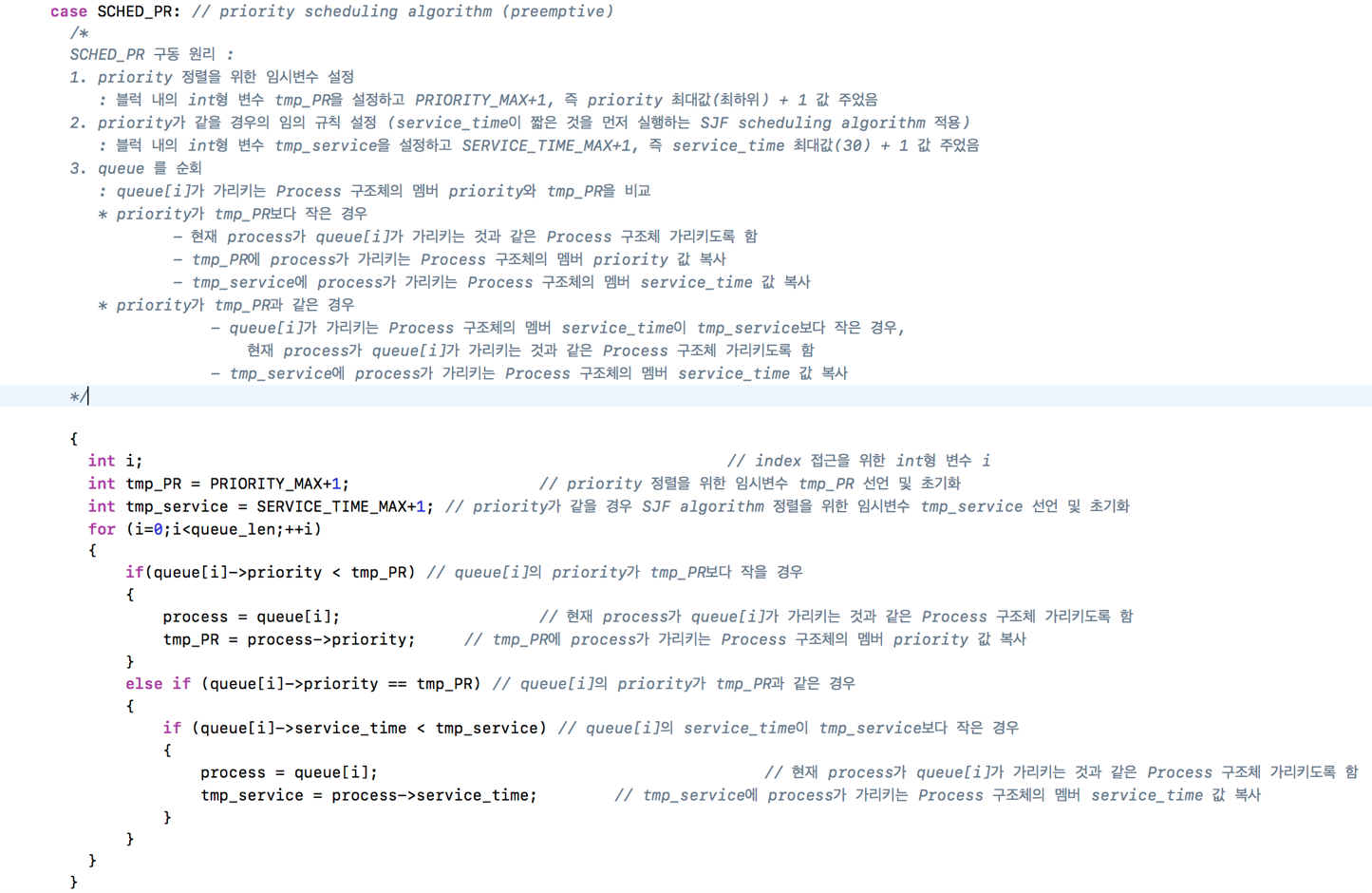
- tmp\_service에 process가 가리키는 Process 구조체의 멤버 service\_time 값 복사

\* priority가 tmp\_PR과 같은 경우

- queue[i]가 가리키는 Process 구조체의 멤버 service\_time이 tmp\_service보다 작은 경우,

현재 process가 queue[i]가 가리키는 것과 같은 Process 구조체 가리키도록 함

- tmp\_service에 process가 가리키는 Process 구조체의 멤버 service\_time 값 복사



4. 실행화면

