숭실대학교

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 txt 파일을 한 줄씩 읽으면서 프로세스의 id 와 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, PR 총 네 가지로 SJF 를 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 SJF 를 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

가) 소스 분석

① sched.c

- **enum**

SCHED_SJF 는 0, SCHED_SRT 는 1, SCHED_RR 는 2, SCHED_PR 는 3, SCHED_MAX 는 4

- **구조체 _Process**

구조체 Process 로 재정의하여 사용
프로세스의 정보를 멤버로 갖는 구조체로 크게 세 가지 역할로 나눌 수 있다.
하기의 표에서 각 역할별 변수들의 정보를 확인할 수 있다.

역할	데이터 형 (data type)	멤버 변수 명 (variable name)	정보 (information)
프로세스 개수 확인용	int	idx	읽은 순서대로 번호 부여받음 0 부터 반영되며, 프로세스가 추가 될 때마다 1 씩 증가 한 값을 부여받음
	int	queue_idx	큐에 들어가는 번호로, 스케쥴링 알고리즘에 따라 변경 될 수 있음.
프로세스 자체의 상세정보	char	id[ID_MAX+1]	프로세스의 id 를 나타내는 것으로, 널문자('\0') 포함 3 개의 char 원소 가짐
	int	arrive_time	프로세스가 cpu_time 기준으로 큐에 도착하는 시간
	int	service_time	프로세스가 cpu_time 기준으로 실행되는데 필요한 시간
	int	priority	프로세스의 우선순위
프로세스 실행과 종료사이 정보 저장용	int	remain_time	프로세스가 종료될 때까지 남은 시간
	int	complete_time	프로세스가 cpu_time 을 기준으로 작업을 모두 끝낸 시각
	int	turnaround_time	프로세스가 완료될 때까지 쓴 총 시간으로, 실제 실행 시간+대기 시간이 적용된 소요시간
	int	wait_time	도착한 프로세스가 할당 조건에 밀려 preemption 당해 큐로 돌아갔다가 다시 실행될 때까지 걸리는 시간. 즉, 대기시간을 뜻함

- **1 차원 구조체 배열 processes[PROCESS_MAX]**

static 키워드를 사용하여 선언과 함께 생성 후 프로그램이 종료될 때까지 유지되며, 본 프로그램에서는 프로세스가 추가될 때에 순서대로 담아놀 자료구조이다.

- **int 형 변수 process_total**

  static 키워드를 사용하여 선언과 함께 생성 후 프로그램이 종료될 때까지 유지되며, 전역에서 선언하였음으로 기본값은 0 이다. read_config 함수를 실행할 때마다 0 으로 초기화되며, append_process 함수에서 프로세스가 추가될 때마다 후위증가연산자로 1 씩 증가한다.

- **1 차원 구조체 포인터 배열 queue[PROCESS_MAX]**

  static 키워드를 사용하여 선언과 함께 생성 후 프로그램이 종료될 때까지 유지되며, 전역에서 선언하였음으로 기본값은 0 으로 NULL, 즉 아무것도 가리키지 않는 셈이 된다. simulate 함수에서 모든 원소가 NULL 로 초기화 되며, cpu_time 에 따라 프로세스가 도착하는대로 queue 배열의 원소가 해당 프로세스를 가리킬 수 있도록 추가된다.

- **int 형 변수 queue_len**

  static 키워드를 사용하여 선언과 함께 생성 후 프로그램이 종료될 때까지 유지되며, 전역에서 선언하였음으로 기본값은 0 이다. simulate 함수 윗부분에서 0 으로 초기화 되며, 프로세스가 더해질 때마다 후위증가연산자로 1 씩 증가한다. 또한 프로세스가 스케쥴링 알고리즘에서 실행되어 종료 될 때에는 큐 길이가 줄어들기 때문에 프로세스에서 실행할 때 필요한 시간이 끝나면 후위감소연산자로 1 씩 감소한다.

- **2 차원 char 배열 schedule[PROCESS_MAX][SLOT_MAX]**

  static 키워드를 사용하여 선언과 함께 생성 후 프로그램이 종료될 때까지 유지되며, 전역에서 선언하였음으로 기본값은 모든 원소가 0 이다. simulate 함수 윗부분에서 모든 원소가 0 으로 초기화 된다. 프로세스 별로 실행되는 순서를 시각화 하기 위해 최대 대기시간 등을 고려하여 최대 갯수인 프로세스 별로 (30+30)*260 개의 칸을 가진다. 이후 간트 차트로 스케쥴링 알고리즘의 성능을 나타낼 때에 '*' 문자를 넣어 cpu_time 기준으로 실행되는 프로세스를 나타내주는 자료구조이다.

- **함수 strstrip**

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

- **함수 check_valid_id**

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

- **함수 lookup_process**

헤더	사용자 정의 함수
형태	static Process* lookup_process (const char *id)
인수	const char* id
반환	Process *
설명	인수로 받은 const char* id 가 지금까지 추가된 processes 의 원소들인 Process 들, 즉 프로세스의 id 와 같은 것이 있는지 확인하여 일치하는 것이 있는 경우, 해당 processes 의 원소 주소를 반환하고 일치하는 것이 없는 경우는 NULL 을 반환.

- **함수 append_process**

헤더	사용자 정의 함수
형태	static void append_process (Process* process)
인수	Process* process
반환	void (없음)
설명	인수로 받은 process 를 processes 배열에 추가. 실행순서는 다음과 같다. 1) processes 배열의 process_total 번째 원소에 인수의 멤버 변수 값 복사 2) processes 배열의 process_tota 번째 원소의 멤버 idx 에 process_total 값 복사 3) 후위증가연산자로 process_total 값 1 증가 (static 변수로 값 계속 반영)

- 함수 read_config

헤더	사용자 정의 함수
형태	static int read_config (const char *filename)
인수	pid_t pid
반환	int
설명	인수로 받은 filename 으로 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 경우 process 의 멤버에 대입한다. 함수의 실행순서는 크게 다음과 같다. 1) fopen 함수에 'r'(read) 모드, 즉 읽기 전용으로 파일 읽기 (파일 열기 실패한 경우 -1 반환) 2) while 문에 fgets 함수 넣어 아래의 경우가 해당될 때까지 한줄씩 읽고, while 문 안 문장 실행 - '\n ' 만나거나 - line 길이 다 끝날 때까지 3) '#' 문자나 빈 문장이면 **continue** 로 while 의 조건문으로 jump 4) id 확인 및 프로세스 id 에 값 복사 - strstrip 함수 사용해 문장 끝('\0')까지 읽고 반환 - strchr 함수로 ' ' 없으면 -1 반환, goto 문을 사용해 **invalid_line** 으로. - check_valid_id 함수 사용해 유효하지 않은 문자일 경우 MSG 함수 사용해 오류 메시지 띄우고 **continue** 로 while 의 조건문으로 jump - lookup_process 함수 사용해 중복된 이름의 id 일 경우 MSG 함수 사용해 오류 메시지 띄우고 **continue** 로 while 의 조건문으로 jump - strcpy 함수로 해당 process 의 멤버 id 에 strstrip 으로 추출된 문자열 복사. 5) arrive time 확인 및 프로세스 arrive_time 에 값 복사 - strstrip 함수 사용해 문장 끝('\0')까지 읽고 반환 - strchr 함수로 ' ' 없으면 -1 반환, goto 문을 사용해 **invalid_line** 으로. - strtol 함수로 strstrip 으로 추출된 문자열을 long 형, 10 진수로 반환하고, 프로세스 arrive_time 에 값을 복사한다. 단, 복사된 값이 ARRIVE_TIME_MAX 보다 크거나 ARRIVE_TIME_MIN 보다 작거나, processes 배열에 추가되기 바로 전 원소 즉, 직전의 프로세스보다 작은경우(도착시간 오류)는 MSG 함수 사용해 오류 메시지 띄우고 **continue** 로 while 의 조건문으로 jump

	6) service time 확인 및 프로세스 service_time 에 값 복사 - strstrip 함수 사용해 문장 끝('\0')까지 읽고 반환 - strchr 함수로 ' ' 없으면 -1 반환되고 goto 문을 사용해 **invalid_line** 으로. - strtol 함수로 strstrip 으로 추출된 문자열을 long 형, 10 진수로 반환하고, 프로세스 service_time 에 값을 복사한다. 단, 복사된 값이 SERVICE_TIME_MAX 보다 크거나 SERVICE_TIME_MIN 보다 작으면, MSG 함수 사용해 오류 메시지 띄우고 **continue** 로 while 의 조건문으로 jump 7) priority 확인 및 프로세스 priority 에 값 복사 - strstrip 함수 사용해 문장 끝('\0')까지 읽고 반환 - strtol 함수로 strstrip 으로 추출된 문자열을 long 형, 10 진수로 반환하고, 프로세스 service_time 에 값을 복사한다. 단, 복사된 값이 PRIORITY_TIME_MIN 보다 작거나, SERVICE_TIME_MAX 보다 크면, MSG 함수 사용해 오류 메시지 띄우고 **continue** 로 while 의 조건문으로 jump 8) append_process 함수로 현재 process 추가 및 **continue** 9) while 문이 끝나면 fclose 함수로 열어둔 파일 닫기

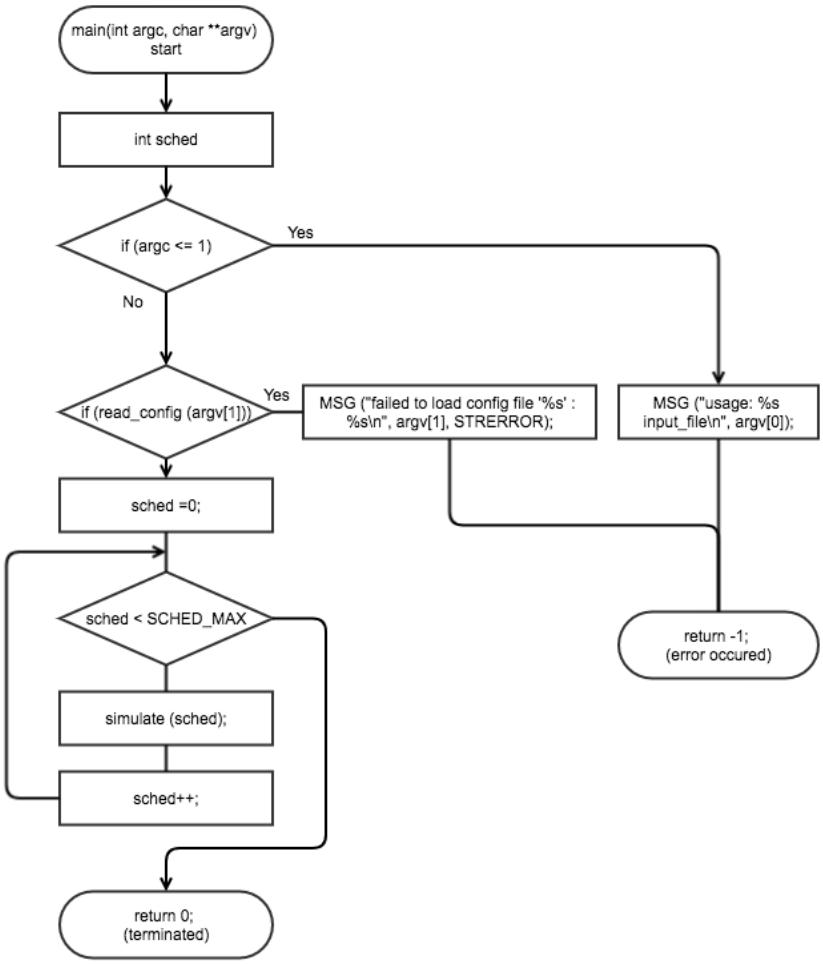
- **함수 simulate**

헤더	사용자 정의 함수
형태	static void simulate (int sched)
인수	int sched
반환	void (없음)
설명	인수로 받은 sched 값에 따라 switch 문으로 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 pp 선언 - pp 가 processes 의 p 번째 원소가 가리키는 것과 같은 Process 가리키도록 주소 대입 - cpu_time pp 의 도착시간이 같지 않으면 반복문 탈출 - pp 가 가리키는 Process 의 remain_time 에 pp 의 service_time 대입 - queue[queue_len] 원소가 pp 와 같은 Process 가리키게 함 - 후위증가연산자로 queue_len 1 씩 증가 3) switch 문에 sched 에 해당하는 값의 case 문 실행 - SCHED_SJF 일 경우 : process 가 NULL 이 아닐 때까지, 현재 shortest 값보다 queue[i]의 멤버 service_time 이 짧은 경우 process 가 queue[i]가 가리키는 Process 를 같이 가리키도록 주소값 복사. shortest 값은 현재 process 의 service_time 대입 - SCHED_SRT 일 경우 : queue_len 이 후위증가연산으로 커지는 I 값보다 작을 때까지 queue[i]원소의 remain_time 이 shortest 보다 작은경우, process 가 queue[i]가 가리키는 Process 를 같이 가리키도록 주소값 복사. shortest 값은 현재 process 의 remain_time 대입 - SCHED_RR 일 경우 : process 를 우선 queue 의 0 번째 원소가 가리키는 Process 를 같이 가리키도록 주소값 복사 i 값이 queue_len-1 보다 작을 때까지 아래의 문장 반복 - queue[i] 원소가 queue[i+1] 원소가 가리키는 Process 를 가리키게 함 - queue[i] 원소의 queue_idx 에 현재 i 값 대입 현재 process 가 가리키는 Process 를 queue[i]도 가리키게 함 queue[i] 원 원소의 queue_idx 에 현재 i 값 대입

- SCHED_PR 은 추가기능 구현에 기재함.
- default 값, 즉 위의 4 개의 값 이외의 값 입력할 경우 MSG 로 오류메시지 출력

4) 스케쥴링할 프로세스가 없을 경우 continue 로 cpu_time 증가 후 다시 반복문 실행
5) schedule 배열의 현재 process 의 idx 번째 값을 행으로, cpu_time 을 열로 가지는 원소에 1 대입
6) 현재 process 의 remain_time 을 후위감소연산자로 1 씩 감소시킴
7) 현재 process 의 remain_time 이 0 보다 작거나 같은 경우, 현재 process 가 완료된 것이므로, 다음 queue 를 실행시키도록 한다. 이 때, 대기열은 줄어든 것이므로 queue_len 은 후위감소연산자로 1 씩 감소시킨다.
8) 현재 process 의 complete_time 즉, 완료시각은 cpu_time 에 1 을 더한 값을 대입
9) 현재 process 의 turnaround_time 즉, 소요시간은 현재 process 의 complete_time 에서 현재 process 의 arrive_time 을 뺀 값을 대입.
10) 현재 process 의 wait_time 은 현재 process 의 소요시간(turnaround_time) – 현재 process 의 도착시각(arrive_time)한 값을 대입
11) 현재 process 를 NULL, 즉 아무것도 가리키지 않도록 설정하고, 완료된 프로세스에 대한 처리가 끝났으므로 후위증가연산자로 p_done 값을 1 씩 증가시키도록 한다.
12) 삼항 연산자로 sched 이 enum 에 명시된 값과 같은 경우 각 스케쥴링 알고리즘 문자열을 출력하게 한다.
13) sum_turnaround_time, sum_waiting_time 값 초기화
14) 간트 차트(Gantt chart)출력 : 이 때도 삼항 연산자로 schedule 배열의 p 행 slot 이 1 이면 '*', 0 이면 ' '문자 출력
15) sum_turunaround_time 에 processes 배열의 p 번째 원소의 turnaround_time 값 더해감 (processes 배열에 추가된 Process 의 turnaround_time 모두 더함)
16) sum_waiting_time 에 processes 배열의 p 번째 원소의 waiting_time 값 더해감 (processes 배열에 추가된 Process 의 waiting_time 모두 더함)
17) avg_turnaround_time 으로 소요시간 합 / 추가된 프로세스 갯수 로 나누고 float 로 캐스팅된 값 대입
18) avg_waiting_time 으로 대기시간 합 / 추가된 프로세스 갯수로 나누고 float 로 캐스팅된 값 대입
19) printf 함수로 cpu_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. terminal 에서 프로그램 실행과 읽어올 txt 파일을 띄어쓰기로 구분하여 읽게 하였는데, 프로그램 실행 명령어만 적을 경우는 오류메시지 띄우고 -1 을 반환하여 비정상 종료되도록 하였다. 이 때, read_config 함수를 이용하였는데, 읽기 실패한 경우에만 MSG 로 오류메시지를 띄우고 -1 을 반환하여 비정상 종료 . sched 가 SCHED_MAX 의 값보다 작을 때까지 simulate 함수 실행 하며 simulate 의 실인자는 sched 가 되도록 하였다. 

나) 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												

② 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, +1 의 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												

③ 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												

④ 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												

다) API 함수 설명

① strlen 함수

헤더	string.h
형태	size_t strlen(const char *s)
인수	const char *s
반환	size_t (문자열의 길이)
설명	문자열 s 의 길이를 계산한다. 이 때, NULL 문자의 길이는 포함되지 않는다.

② strchr 함수

헤더	string.h
형태	char* strchr(const char* s, int c)
인수	const char *s
반환	char*(문자열 s 에서 발견된 문자 c 의 포인터)
설명	문자 배열 s 내에 문자 c 가 있는지 검사하고 있을 경우 문자 c 가 있는 번지를 반환한다. 만약 문자열 s 에 문자 c 가 발견되지 않을 경우 NULL 을 리턴한다.

③ strcmp 함수

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

④ strcpy 함수

헤더	string.h
형태	char *strcpy(char *dest, const char* src);
인수	char *dest, const char* src
반환	char* (dest+strlen(src)의 번지 반환)
설명	문자열 src 를 문자배열 dest 로 복사한다. NULL 문자도 함께 복사가 된다. dest 크기가 src 크기보다 같거나 커야 복사가 온전히 일어나며, 만일 dest 문자열 크기가 src 문자열 크기보다 작은 경우 dest 뒤의 인접데이터가 손실된다.

⑤ strtol 함수

헤더	stdlib.h
형태	long strtol (const char*s, char** endptr, int radix)
인수	const char*s, char** endptr, int radix
반환	long (변환된 long 형 정수 반환)
설명	인자로 받은 문자열 s 을 radix 로 받은 진법으로 전환하여 long 형 값으로 반환한다. 이 때 불필요한 문자가 문자열사이에 있거나, 문자열이 [공백,tab][부호][숫자] 순서로 나와있지 않은 경우는 변환이 중지되고, 중지된 위치를 endptr 에 설정해준다. 따라서 endptr 은 디버깅시 이용할 수 있다. 본 프로그램에서는 10 진수로 사용하였는데, radix 가 0 인 경우는 숫자 형태에 따라 진법을 구분한다. 0abc 인 경우는 8 진수로, 0xabc 일 때는 16 진수로, abcdlf EOsms 10 진수로 인식하게 된다.

⑥ isspace 함수

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

⑦ isupper 함수

헤더	ctype.h
형태	int isupper(int c)
인수	int c
반환	int (c 가 A~Z 이면 0 이외의 값(참) 리턴, 아니면 0(거짓) 반환)
설명	문자 c 가 대문자인지를 검사해준다. 그 외의 문자일 경우는 0 을 반환한다

⑧ isdigit 함수

헤더	ctype.h
형태	int isdigit(int c)
인수	int c
반환	int (c 가 0~9 사이의 문자면 0 이외의 값(참) 리턴, 그 외의 문자면 0(거짓) 반환)
설명	문자 c 가 숫자인지를 검사해준다. 그 외의 문자일 경우는 0 을 반환한다

⑨ memmove 함수

헤더	string.h
형태	void* memmove (void *dest, const void*src,, size_t n);
인수	int c
반환	void* (dest 번지를 반환하며, 실패하면 NULL 포인터 반환)
설명	src 번지에 있는 데이터를 dest 가 지정하는 번지로 n 바이트만큼 복사 str 번지에 있는 데이터를 start 가 지정하는 번지로, src+(src 의 데이터형인 char 형(1byte) *1)바이트만큼 복사

⑩ memset 함수

헤더	string.h
형태	void* memset (void *dest, int c, size_t n);
인수	void *dest, int c, size_t n
반환	void* dest (dest 번지)
설명	보통 할당받은 메모리를 특정 값(c)으로 초기화 하는 것으로, NULL 로 초기화 할 때에 많이 사용. 디버깅에 유리함. dest 가 올바르지 않은 경우 NULL 포인터 반환

⑪ fgets 함수

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

⑫ fopen 함수

원형

FILE *fopen(const char *filename,const char *mode);
⊙ filename : 개방하고자 하는 파일 이름, 경로 포함 가능
⊙ mode : 개방 방식, 액세스 모드, 파일 형태 지정

헤더 파일

stdio.h

기능

스트림을 개방하고 FILE형 구조체를 만들어 개방한 스트림에 관한 정보를 기입한 후 그 포인터를 리턴한다.
리턴된 스트림 포인터는 그 후 fgetc, fputc 등의 스트림 I/O 함수에 의해 사용된다. filename은 개방하고자 하는 파일의 이름이며 필요시에는 드라이브, 디렉토리 경로를 포함할 수 있다. 경로(path)를 포함할 경우에 역슬래시 기호는 확장열로 \\임을 주의하자.
다음과 같은 filename이 타당하다.

"MYLOVE.EXE"
"C:\\KSH\\BABO\\DA.TXT"

mode는 파일에 어떤 조작을 가할 것인지(파일 액세스 모드)와 파일 형태를 지정한다. 파일 액세스 모드에는 다음과 같은 것들이 있다.

r : 읽기 전용으로 기존의 스트림을 개방한다. 이 모드로 개방된 파일에 쓰기를 하는 것은 불가능하다. 스트림이 존재하지 않으면 에러가 발생한다.
w : 쓰기 전용으로 새로운 스트림을 만든다. 이 모드로 개방된 파일을 읽을 수는 없다. 새로 스트림을 만드는 것이기 때문에 파일은 기존에 존재하지 않아야 한다. 만약 존재한다면 기존의 파일은 지워지게 된다.
a : 추가를 위해 스트림을 개방한다. 추가란 스트림의 뒷부분에 무엇인가 다른 정보를 기입한다는 뜻이며 이 모드로 개방된 스트림의 FP는 추가를 위해 파일의 끝으로 이동된다. 만약 스트림이 존재하지 않는다면 스트림을 만든다. 이 경우는 w와 동일해진다.
r+, w+ : 읽고 쓰는 것이 다 가능한 상태로 개방이 된다. 이 모드로 개방된 스트림은 입출력이 다 가능하다. r+와 w+의 차이는 스트림이 존재하지 않을 경우 r+는 에러를 내고 리턴하며 w+는 새로운 스트림을 만든다는 것이다.
a+ : 추가를 위해 스트림을 개방한다. 추가란 스트림의 뒷부분에 무엇인가 다른 정보를 기입한다는 뜻이며 이 모드로 개방된 스트림의 FP는 추가를 위해 파일의 끝으로 이동된다. 만약 스트림이 존재하지 않는다면 스트림을 만든다. 이 경우는 w+와 동일해진다.

파일의 형태란 개방하고자 하는 파일의 형태를 지시해 주는 것이다. 개방할 파일이 텍스트 파일이면 "t", binary 파일(이진 파일)이면 "b"를 붙여준다.

다음은 타당한 모드의 예이다.

"rt" : 텍스트 파일을 읽기 전용으로 개방. 파일의 리스트를 출력할 때 적당하다.
"wb" : 이진 파일을 쓰기 전용으로 개방. 파일 복사시에 적당하다.
"r+b" : 이진 파일을 갱신용으로 개방.
"r" : 읽기 위해서 연다. 파일의 형태는 전역변수 _fmode에 따른다.

만약 mode에 t나 b 등의 파일 형태를 명시하지 않을 경우 fcntl.h에 정의되어 있는 전역변수 _fmode의 값을 사용하는데 디폴트는 O_TEXT로 설정되어 있어 t를 붙인 것으로 간주된다.

리턴값 개방된 스트림의 포인터(FILE *)를 리턴한다.
에러가 발생한 경우 NULL을 리턴한다. 파일을 개방한 후에는 반드시 리턴값을 점검하여 에러 처리를 해 주어야 한다.

⑬ fclose 함수

원형 int fclose(FILE *stream);
⊙ stream : 닫고자 하는 스트림

헤더 파일 stdio.h

기능 개방된 스트림을 닫는다. 버퍼에 남아 있는 출력용 데이터는 모두 파일로 기록되고 입력용 데이터는 모두 지워진다. 즉 버퍼가 비워진다.
스트림에 할당되어 있는 버퍼는 해제되지만 setbuf나 setvbuf로 지정된 버퍼는 자동으로 해제되지 않으므로 직접 해제해 주어야 한다.

리턴값 성공하면 0을 리턴하고 에러가 발생한 경우 EOF(-1)을 리턴한다. 스트림이 유효하면 에러가 발생할 확률은 지극히 낮으므로 에러 점검은 꼭 하지 않아도 상관이 없다.

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

2) 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

3) 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_service 에 process 가 가리키는 Process 구조체의 멤버 service_time 값 복사

```
case SCHED_PR: // priority scheduling algorithm (preemptive)
  /*
  SCHED_PR 구동 원리 :
  1. priority 정렬을 위한 임시변수 설정
     : 블럭 내의 int형 변수 tmp_PR을 설정하고 PRIORITY_MAX+1, 즉 priority 최대값(최하위) + 1 값 주었음
  2. 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
  3. 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
  */

  {
    int i;                                                  // index 접근을 위한 int형 변수 i
    int tmp_PR = PRIORITY_MAX+1;                    // priority 정렬을 위한 임시변수 tmp_PR 선언 및 초기화
    int tmp_service = SERVICE_TIME_MAX+1; // priority가 같을 경우 SJF algorithm 정렬을 위한 임시변수 tmp_service 선언 및 초기화
    for (i=0;i<queue_len;++i)
    {
        if(queue[i]->priority < tmp_PR) // queue[i]의 priority가 tmp_PR보다 작을 경우
        {
            process = queue[i];                     // 현재 process가 queue[i]가 가리키는 것과 같은 Process 구조체 가리키도록 함
            tmp_PR = process->priority;     // tmp_PR에 process가 가리키는 Process 구조체의 멤버 priority 값 복사
        }
        else if (queue[i]->priority == tmp_PR) // queue[i]의 priority가 tmp_PR과 같은 경우
        {
            if (queue[i]->service_time < tmp_service) // queue[i]의 service_time이 tmp_service보다 작은 경우
            {
                process = queue[i];                                     // 현재 process가 queue[i]가 가리키는 것과 같은 Process 구조체 가리키도록 함
                tmp_service = process->service_time;        // tmp_service에 process가 가리키는 Process 구조체의 멤버 service_time 값 복사
            }
        }
    }
  }
```

4. 실행화면

```
jihyunkang@jihyunkang-VirtualBox:~/Downloads/os_hw2/test1$ ./sched data1.txt
duplicate process id 'P3' in line 11, ignored
invalid arrive-time '-1' in line 14, ignored
invalid arrive-time '31' in line 15, ignored
invalid arrive-time '0' in line 18, ignored
invalid arrive-time '0' in line 19, ignored
invalid priority '0' in line 22, ignored
invalid priority '11' in line 23, ignored
invalid priority '-1' in line 24, ignored

[SJF]
P1 ***
P2    ******
P3           ****
P4               *****
P5          **
CPU TIME: 20
AVERAGE TURNAROUND TIME: 7.60
AVERAGE WAITING TIME: 3.60

[SRT]
P1 ***
P2    *      *****
P3     ****
P4               *****
P5         **
CPU TIME: 20
AVERAGE TURNAROUND TIME: 7.20
AVERAGE WAITING TIME: 3.20

[RR]
P1 ***
P2    ** *  *   *   *
P3      * *  *   *
P4         *   *   * **
P5           *   *
CPU TIME: 20
AVERAGE TURNAROUND TIME: 10.40
AVERAGE WAITING TIME: 6.40

[PR]
P1 ***
P2    *      *****
P3     ****
P4               *****
P5         **
CPU TIME: 20
AVERAGE TURNAROUND TIME: 7.20
AVERAGE WAITING TIME: 3.20
jihyunkang@jihyunkang-VirtualBox:~/Downloads/os_hw2/test1$
```