OS Assignment#2

운영체제 스케줄러 알고리즘 시뮬레이션

학부 : 소프트웨어 학부

학번 : 20150291

이름 : 하현수

1. 소개

과제 개요

본 과제는 다양한 운영체제 스케줄러 알고리즘을 사용하여 SJF,SRT,RR,PR 방식의 프로세스 할당 상태를 보여주는 sched.c라는 프로세스 스케줄러 프로그램의 소스 코드를 추가 구현하여 미구현된 PR 스케줄링 알고리즘을 구현하는 것을 목표로 합니다.

case문의 PR부분에 해당하는 For 루프문을 추가하여 PR 스케줄링 알고리즘을 성공적으로 추가 구현하였습니다.

구현에 대한 테스트는 Vmware Workstation 12 버전을 통한 우분투 16.04 가상머신을 통해 진행하였습니다.

관련 연구는 스케줄링 알고리즘에 대한 조사와 사용된 API함수 조사로 구성되어 있으며, 스케줄링 알고리즘에 대한 조사 부분에서는 스케줄링 알고리즘 기법의 종류에 대한 내용과, 예제 파일에서 사용된 스케줄링 알고리즘 및 그 외에도 각종 보완 알고리즘 등을 담았습니다.

API함수 부분에서는 사용된 API함수를 다룹니다. 저번 과제와 기본적인 뼈대가 비슷하여, 저번 과제에서 이미 분석한 API함수는 저번과제와 동일하게 작성하였습니다.

Makefile의 test부분이 result 텍스트 파일에 기록되지 않고 무한루프가 발생하는 문제가 있어, &>를 >로 수정하였습니다.

2. 관련 연구

스케줄링 기법의 종류

스케줄링 기법은 진행중인 프로세스에서 자원을 뺏고 다른 것을 실행할 수 있는지의 여부에 따라서 선점(Preemptive) 기법과 비선점(Non-Preemptive)기법으로 나뉩니다.

프로세스가 CPU를 점유하고 있을 때 다른 프로세스가 프로세서의 자원을 빼앗을 수 있는 방법을 선점 스케줄링이라고 칭합니다. 선점 스케줄링은 우선 순위가 높은 프로세스가 CPU를 먼저 차지하기가 용이하기 때문에 실시간 시분할 시스템에서 사용됩니다.
우선 순위가 높은 프로세스가 먼저 수행되어야 할 때 유용하며, 빠른 응답시간을 요구하는 대화식 시분할 시스템이나 처리 시간이 제한되어 있는 실시간 시스템에 유용합니다.
단점은 많은 오버헤드를 초래한다는 점입니다.
대표적인 선점 스케줄링은 RR, SRT, MFQ등이 있습니다.

프로세스에게 이미 할당된 CPU를 강제로 빼앗을 수 없고, 그 프로세스의 사용이 끝난 후에만 다음 프로세스 할당이 가능한 스케줄링 방식을 비선점 스케줄링이라고 힙니다.
중간 중지가 없이 이루어지기 때문에, 모든 프로세스에 대한 요구를 공정하게 처리할 수 있고, 응답 시간의 예측이 용이합니다. 하지만, 짧은 작업이 긴 작업을 기다리기 위해 낭비하는 시간이 발생한다는 단점이 있습니다.

스케줄링 알고리즘

FIFO(First In First Out) 스케줄링

FIFO는 가장 간단한 스케줄링 기법으로, 먼저 대기 큐에 들어온 작업에게 CPU를 먼저 할당하는 비선점 방식의 스케줄링입니다. 중요하지 않은 작업이 중요한 작업보다 우선적으로 처리될 수 있으며, 응답 시간이 늦어 대화식 시스템에 부적합합니다.
FCFS(First Come First Served)라고 불리기도 합니다.

SJF(Shortest Job First) 스케줄링

SJF는 비선점 스케줄링 기법에 속하는 스케줄링 기법으로, 처리해야 할 작업 시간이 가장 작은 프로세스에 CPU를 할당하는 방식입니다. 평균 대기 시간이 최소인 최적의 알고리즘이지만, 각 프로세스의 CPU 요구 시간을 미리 알기 어렵다는 단점을 가지고 있습니다.

SRT(Shortest Remaining Time First) 스케줄링

SRT는 SJF 스케줄링 기법을 선점 방식으로 구현한 방식입니다. 새로 도착한 프로세스를 포함하여, 대기 큐에 남아 있는 프로세스의 작업이 완료되기까지의 남아 있는 실행 시간이 가장 적은 프로세스에 CPU를 할당합니다.

RR(Round Robin) 스케줄링

RR은 FIFO 스케줄링 기법을 선점 방식으로 구현한 방식입니다. 프로세스가 FIFO 형태로 대기 큐에 올라가지만, 주어진 시간(Time Slice)안에 작업을 마쳐야 하며, 할당량을 다 소비했는데도 작업이 끝나지 않았다면 대기 큐의 맨 뒤로 되돌아가게 됩니다.
시스템이 사용자에게 적절한 응답시간을 제공해 주는 대화식 시분할 시스템에 적합합니다.

PR(Priority) 스케줄링

PR은 각 작업마다 우선순위를 주어준 뒤, 우선 순위가 높은 작업 순으로 CPU를 할당하는 방식입니다. 우선순위가 낮은 프로세스가 Starvation에 빠질 수 있고, 이를 해결하기 위해서, 체류 시간이 길어질수록 우선순위를 높여주는 Aging 기법을 이용합니다.

Deadline 스케줄링

Deadline 스케줄링은 작업이 주어진 제한 시간이나 Deadline 안에 완료되도록 하는 기법입니다.

HRN(Highest Response Ratio Next) 스케줄링

HRN은 Brinch Hansen이 구현한 알고리즘으로, SJF 스케줄링의 단점인 긴 작업과 짧은 작업의 지나친 불평등 현상을 보완한 스케줄링 기법입니다.
비선점 스케줄링 기법이며, 서비스 받을 시간이 분모에 있으므로 짧은 작업의 우선 순위가 높아지는 방식이며, 대기 시간은 분자에 있으므로 긴 작업도 대기 시간이 클 경우에는 우선 순위가 높아지는 방식을 이용하여 효과적인 스케줄링을 구현한 방식입니다.

MFQ(Multi-level Feedback Queue)스케줄링

MFQ는 다양한 특성의 프로세스가 혼합된 경우 매우 유용하게 쓰이는 스케줄링 방식으로 새로운 프로세스를 그 특성에 따라 각각 대기 큐에 삽입하고, 그 실행 형태에 따라 다른 대기 큐로 이동시킵니다. 예시로, 연산 위주의 프로세스들을 처음에 RR방식의 대기큐에서 주어진 시간 할당량이 만료되면 다음 단계에 큐에 배치하고, 실행 시간이 길수록 점점 낮은 우선순위를 지니게 되어 마지막 가장 낮은 우선 순위의 큐에 도달하면 작업이 끝날 때 까지 RR 방식으로 스케줄링합니다.

사용된 API 함수 연구

memmove() 함수: memcpy() 처럼 메모리 영역을 복사하는 함수입니다. 다른 점은 하나의 포인터에 대해서 동일한 영역 내에서 복사가 가능합니다. string.h 헤더 파일에 존재합니다.

strlen() 함수 : 문자의 길이를 판별하는 함수입니다. 널 문자를 제외한 String의 길이를 출력합니다. 헤더파일은 string.h 입니다.

isspace() 함수 : 문자가 공백 문자인지 판별하는 함수입니다. 헤더파일은 ctype안에 있으며, 공백이 아닐 경우 0을 리턴합니다.

isupper() 함수 : 문자가 대문자인지 판별합니다. 헤더파일은 ctype.h 이며, 대문자가 아닐 경우 0을 리턴합니다.

isdigit() 함수 : 문자가 숫자인지 판별합니다. 헤더파일은 ctype.h 이며, 숫자가 아닐 경우 0을 리턴합니다.

strcmp() 함수 : 문자열을 비교합니다. 헤더파일은 string.h 이며, 같을 경우 0를 리턴합니다.

fgets() 함수 : 파일 스트림으로부터 문자열을 받아 옵니다. 헤더파일은 stdio.h이며, 정상적으로 읽기를 수행했다면 메모리 포인터를 반환하며, 파일 끝이거나 문제가 발생했을 경우 NULL을 반환합니다.

fopen() 함수 : open() 시스템콜과 달리 내부적으로 라이브러리를 이용한 Library Call 함수입니다. 파일을 열고 해당 파일 포인터를 리턴합니다. 헤더파일은 stdio.h 입니다.

memset() 함수 : 할당 받은 메모리를 특정 값으로 초기화 하는 함수입니다.
메모리 관련 함수 이지만 헤더 파일은 string.h 파일입니다.

strchr() 함수 : 문자열에서 찾는 문자에 해당하는 포인터 값을 리턴하는 함수 입니다. 헤더파일은 string.h 입니다.

strcpy() 함수 : 문자열을 복사하는 함수입니다. 복사가 완료된 포인터 값을 리턴하며 헤더파일은 string.h 입니다.

strtol() 함수 : 숫자 분자열을 long형 숫자로 변환하는 함수입니다. atoi나 atol과는 달리 변환하려는 진수를 정할 수 있고, 숫자가 아닌 문자열을 만나면 그 포인터의 위치를 구할 수 있습니다. 헤더파일은 stdlib.h이며, 10진 long형 정수 값을 반환합니다.

fclose() 함수 : fopen으로 열은 파일을 닫는 파일 포인터입니다. 헤더파일은 stdio.h이며, 정상적으로 닫았다면 0을, 오류가 발생하면 EOF를 반환합니다. close() 시스템콜과 다른, 라이브러리 콜 함수입니다.

putchar() 함수 : 표준 출력 장치로 문자 하나를 출력하는 함수입니다. 헤더파일은 stdio.h이며, 출력한 문자를 반환합니다.

printf() 함수 : 서식에 맞추어 표준 출력 장치로 출력합니다. 다양한 출력 서식을 지정하여 사용할 수 있습니다. 헤더파일은 stdio.h이며, 출력된 문자 수를 반환합니다. 만약 오류 발생시 음수를 반환합니다.

3. 추가 기능 구현 방법

구현 방법

TO BE IMPLEMENTED 주석 부분에 해당하는 SCHED_PR 케이스에 for문을 작성하였습니다.

```
case SCHED_PR:
    /* TO BE IMPLEMENTED */
{
    int i;//for문을 위한 변수 선언
    int best_priority = PRIORITY_MAX + 1;//가장 먼저 처리해야할 우선순위 변수를 선언하고, 최대값+1로 초기화합니다.
    process = queue[0];
    for (i = 0; i < queue_len; i++)
    {
        if (queue[i]->priority < best_priority)//최초 실행시 최대값+1의 우선순위보단 당연히 처음 들어오는 priority가 작으므로 if문에 걸립니다.
        {
            process = queue[i];
            best_priority = process->priority;
        }//우선순위가 가장 높다면, 현재 프로세스를 변경하고 best_priority를 자신의 priority로 설정합니다.
    }
}
    break;
```

for문을 위한 변수 i를 선언하고

best_priority라는 우선순위 비교 변수를 선언하였습니다. 최초값은 우선순위의 최대 값인 10(PRIORITY_MAX)에 1을 더한값인 11로 설정합니다.

최초 if문 실행시 대기 큐에 들어온 첫 프로세스는 반드시 best_priority보다 priority가 높으므로, 먼저 들어온 프로세스가 실행되고, 해당 프로세스의 prioirty가 best_prioirity로 지정됩니다.

만약 새로 들어온 프로세스가 현재 프로세스보다 우선순위가 높다면, 선점을 통해 우선순위가 높은 프로세스를 먼저 실행 하게 됩니다.

또한 만약 프로세스의 우선순위가 같을 경우에는, 이미 실행되고 있던 프로세스를 마친 다음에 들어온 프로세스에 자원을 할당하도록 하였습니다.

실행 결과

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

실행결과 위와 같은 결과가 나왔습니다.

사용한 데이터 파일은

```
P1 0 3 1
P2 2 6 2
P3 4 4 1
P4 6 5 3
P5 8 2 1
```

로 이루어진 기본적으로 주어진 data1.txt입니다.

우선 P1을 실행하고, P2를 실행하다가, 대기 큐에 들어온 P3의 우선순위가 더 높음을 확인하고 P3를 진행한 후, 대기 큐에 있는 동일 우선순위의 P5를 실행합니다. 그 후, 우선순위 2의 P2를 다시 진행한 후, 우선순위 3인 P4로 다시 돌아 갑니다.

실행 결과 자체는 논리적으로 오류가 없지만, 간트 차트의 모양이 SRT와 완전히 똑같고, 턴어라운드 평균 시간과 평균 대기 시간도 일치하는것으로 나타났습니다.

그래서 보다 직접적인 확인을 위해서 data파일의 우선순위를 멋대로 변경해보았습니다.

```
P1 0 3 3
P2 2 6 1
P3 4 4 2
P4 6 5 3
P5 8 2 1
```

위와 같이 data 파일의 우선순위를 변경하여 실행시켜보겠습니다.(temp_data.txt)

```
[PR]
P1 **                *
P2    ******
P3              ****
P4                    *****
P5           **
CPU TIME: 20
AVERAGE TURNAROUND TIME: 9.40
AVERAGE WAITING TIME: 5.40
```

위와 같은 결과를 얻을 수 있었습니다.

우선 순위 3인 P1이 우선 대기 큐에 들어와 실행됩니다.

그러던중 우선순위가 1인 P2가 대기 큐에 들어오면, 우선순위가 더 높은 P2가 우선적으로 실행됩니다. 그 후엔 동일한 우선순위를 가지는 P5를 실행시키고,

우선 순위가 2인 P3를 실행합니다. 그 후, 먼저 작업을 진행했었던 P1을 다시 시작하여 마치고, P4를 처리합니다.

논리적으로 오류가 없으면서, 우선순위를 따라서 잘 작동하는 것을 확인 할 수 있었습니다.

이상 과제 보고서를 마치겠습니다.

감사합니다!