운영체제 프로그래밍 과제

< CPU Scheduling Simulator >

이번 프로그래밍 과제 수업 시간에 배운 프로세스의 상태전환(Process State Transitions)과 CPU Scheduling에서 여러 가지 Scheduling 기법을 구현하여 시뮬레이션 해 보는 것입니다. 실제 process 가 수행하는 것을 정확히 시뮬레이션 하는 것은 여러 가지 어려움이 있기에 CPU Scheduling을 간단한 프로세스 수행 모델 (Process Execution Model) 하에서 구현 합니다.

Process Execution Model

시뮬레이션 동안 생성될 프로세스의 수행시간, 프로세스의 생성 시간, IO 요청이 발생하는 시간과 IO 요청의 처리 시간은 시뮬레이션이 시작하기에 정해지면 각각 다음의 배열에 저장됩니다. procServTime[], procIntArrTime[], ioReqIntArrTime[], ioServTime[].

가) 프로세스의 생성 시간

실제에서는 프로세스가 수행 중 다른 프로세스를 만듭니다(fork). 하지만 이 모델에서 각 프로세스는 정해진 시간에 만들어집니다. 이는 프로세스 가 만들어지고 다음 프로세스가 만들어지는 시간으로 표현합니다 (Process Interarrival Time). 시뮬레이션 코드에서는 procIntArrTime[] 배열에 시간이 저장됩니다. 첫 번째 프로세스는 시뮬레이션이 시작하고 procIntArrTime[0]에 기록된 시간 후에 생성됩니다. 두 번째 프로세스는 첫 번째 프로세스가 만들어지고 procIntArrTime[1]에 기록된 시간 뒤에 생성됩니다. 즉 N 번째 프로세스는 N-1 번째 프로세스가 만들어지고 procIntArrTime[N-1] 에 기록된 시간 뒤에 생성됩니다.

나) 프로세스의 수행시간

프로세스는 정해진 수행시간 만큼 CPU를 사용하고 종료하게 됩니다. 프로세서의 수행시간은 MIN ~ MAX 값 사이에서 램덤하게 형성됩니다. N 번째 프로세스의 수행 시간은 procServTime[N-1]에 저장되고 시뮬레이션이 시작되기 전에 프로세스의 (procTable[N-1]) targetServiceTime의 필드에 저장됩니다.

다) 프로세스의 IO 요청 시간

실제에서는 프로세스가 수행하며 직접 IO를 요청하지만 이 모델에서는 IO 요청도 프로세스와 비슷하게 정해진 시간에 만들어집니다. 이는 IO 요청이 만들어지고 다음 IO 요청이 만들어지는 시간으로 표현 됩니다 (IO Request Interarrival Time). 시뮬레이션 코드에서는 ioReqIntArrTime[] 배열 시간이 저장됩니다. IO 요청은 프로세스가 CPU 사용하는 중간에만 발생하기 때문에 IO Request Interarrival Time은 시뮬레이션 시간이 아닌 CPU 사용 시간을 기준으로 사용합니다. 따라서 첫 번째 IO Request는 CPU 사용시간이 시작되고 ioReqIntArrTime[0]에 기록된 CPU 사용 시간 후에 발생합니다. 두 번째 IO Request는 첫 번째 IO Request가 발생하고 CPU 사용 시간이 ioReqIntArrTime[1]이 기록된 시간 만큼 지난 후에

발생됩니다. 즉 N 번째 IO Request는 N-1 번째 IO Request 가 만들어지고 CPU 가 ioReqIntArrTime[N-1] 에 기록된 시간만큼 사용된 뒤에 생성됩니다.

라) IO 처리 시간
IO 요청이 발생하고 해당 IO를 처리하고 완료하는데 걸리는 시간을 나타냅니다. 각 IO의 처리시간은 MIN ~ MAX 값 사이에서 램덤하게 형성됩니다. N 번째 IO 요청에 대한 처리 시간은 ioServTime[N-1]에 저장됩니다.

CPU Scheduling System Model

모든 프로세스의 상태는 procTable[] 에 저장됩니다. 프로세스가 생성되는 순서대로 해당 배열에 차례로 할당됩니다. procTable[] 은 프로세스의 연산, 스케쥴링을 위한 정보, 통계 정보 등을 위한 모든 데이터를 저장합니다.

프로세스의 상태는 S_IDLE, S_RUNNING, S_READY, S_BLOCKED, S_TERMINATE 로 나누어집니다. CPU를 사용할 프로세스가 없는 경우 idle 프로세스(idleProc)가 CPU를 사용합니다. 이 프로세스의 상태는 S_IDLE로 시작합니다. 시뮬레이션 동안 idle 프로세스의 상태는 중요하지는 않습니다. Scheduler에 의해 선정된 프로세스가 CPU를 사용하게 되면 해당 프로세스의 상태는 S_RUNNING로 바뀌고 CPU register들 (cpuReg0, cpuReg1)의 값 을 복원하고 매 사이클 compute() 호출하여 CPU register들을 사용하여 연산을 합니다. 시뮬레이션이 진행하는 동안에 다음과 같은 이벤트가 발생합니다.

가) 프로세스의 생성
새로이 생성된 프로세스는 Ready Queue에 들어갑니다. 프로세스의 생성은 CPU가 idle 상태, 즉 idle 프로세스가 CPU 사용하는 중에도 발생할 수 있습니다. 현재 Running 중인 프로세스는 새로이 생성된 프로세스 다음에 Ready Queue에 첨가됩니다. 그리고 새로운 프로세스가 생성되었으므로 Scheduler를 호출하게 됩니다.

나) Quantum 만료
Quantum 만료된 경우에는 현재 Running 프로세스는 Ready Queue로 가고 Scheduler가 호출되어 다음에 CPU를 사용한 프로세스를 선정합니다. idle 프로세스가 CPU를 사용하는 동안에 quantum 만료가 일어나도 큰 의미는 없습니다.

다) IO 요청
Running 프로세스가 있는 경우 IO 요청이 발생할 수 있습니다. 이 경우 해당 Running 프로세스가 IO 요청을 하는 것으로 처리합니다. IO 요청을 한 프로세스는 Blocked 상태로 전환됩니다. 그리고 Scheduler가 호출되어 다음에 CPU를 사용할 프로세스를 선정합니다. 시뮬레이션에서는 N번째 IO 요청이 발생할 경우 해당 요청이 끝나는 시간을 ioServTime[N-1]에서 가져와 ioDoneEvent를 만들어 (ioDoneEvent[N-1]) ioDoneEventQueue에 더하게 됩니다. IO 요청이

같은 시간에 끝날 경우 나중에 들어온 ioDoneEvent가 먼저 들어온 ioDoneEvent 뒤에 삽입됩니다.

라) IO 처리가 끝남 (IO Done Event)
IO 처리는 프로세스의 생성과 마찬가지로 Running 프로세스가 있거나 CPU가 idle 하거나 상관없이 특정 IO 요청에 대한 처리시간이 지나면 발생합니다. IO 처리가 끝나는 경우 해당 IO를 요청했던 프로세스는 Blocked에서 Ready상태로 전환하게 됩니다. 단 이때 해당 프로세스가 이미 수행을 마쳤으면(S_TERMINATE) Ready로의 상태전환은 필요 없습니다. Running 프로세스가 있는 상황에서 IO Done Event 가 발생하면 Running 프로세스는 Ready 상태로 전환되고 Scheduler가 호출되어 다음에 CPU를 사용할 프로세스를 선정합니다. Running 프로세스의 Ready 상태 전환이 IO Done Event에 의한 Ready 상태 전환 뒤에 이루어집니다. IO Done Event 가 이미 종료된 프로세스의 것이라도 같은 방식으로 처리합니다. 같은 시뮬레이션 시간에 여러 개의 IO 요청이 동시에 만료 가능합니다.

마) 종료(Terminate)
Running 프로세스의 CPU를 사용한 시간(serviceTime)이 지정된 프로세스의 수행시간(targetServiceTime)에 도달하면 해당 프로세스는 수행을 중단하고 S_TERMINATE 상태가 됩니다. Scheduler가 호출되어 다음에 CPU를 사용할 프로세스를 선정합니다.

가) ~ 마)에 해당하는 이벤트는 동시에 발생할 수 있습니다. 이 경우 Running 프로세스의 상태 전환이 일치 하지 않은 경우에는 그 상황에 맞게 Running 프로세스의 상태를 전환해야 합니다. 예를 들어 IO 요청을 하고 동시에 종료하는 경우 해당 프로세스는 종료상태로 전환됩니다. 즉 다음과 같은 순서로 상태전환이 우선 됩니다. 종료 > Blocked > Ready

각 이벤트의 진행에서 Ready 상태로의 전환이 필요한 여러 경우가 발생할 때 항상 현재 Running 프로세스가 제일 나중에 Ready 상태로 전환됩니다.

모든 프로세스 (NPROC)가 종료되면 시뮬레이션이 종료되고 여러 통계 자료 등을 출력합니다.
최종 통계자료의 출력물은 printResults() 함수에서 출력하는데 프로세스가 시작해서 (생성되어서) 끝날 때까지의 시간의 모든 프로세스의 평균값과 compute() 연산을 통해 얻어진 두 개의 register 값을 더한 모든 프로세스의 평균값이 출력됩니다.

CPU Scheduling Simulator 인자

SCHEDULING_METHOD : CPU scheduling에서 사용할 알고리즘을 나타낸다.
각각의 스케쥴링은 다음과 같은 번호를 갖는다. Round Robin Scheduling = 1, Shortest Job First = 2, Shortes Remaining Time Next = 3, Guaranteed Scheduling = 4, Simple Feedback Scheduling = 5

NPROC : 시뮬레이션에 생성될 총 프로세스의 수

NIOREQ : 시뮬레이션 중 생성될 총 IO 요청의 수

QUANTUM : preemptive 스케쥴링에서 사용될 Quantum 시간

MIN_INT_ARRTIME, MAX_INT_ARRTIME : 프로세스가 생성될 때 사용되는 process interarrival time의 최소, 최대 시간 값

MIN_SERVTIME, MAX_SERVTIME : 각 프로세스에 할당 된 수행시간의 최소, 최대 시간 값

MIN_IO_SERVTIME, MAX_IO_SERVTIME : 각 IO 요청이 발생 했을 때 이 요청이 완료되기 위해 필요한 최소, 최대 시간값

MIN_IOREQ_INT_ARRTIME : IO 요청이 생성되는 시간 간격 중 최소 시간 값.
참고사항: IO request의 interarrival time은 모든 프로세스의 할당된 수행 시간의 총합 (totalCPUTime)을 IO 요청의 총수 + 1 (NIOREQ+1)로 나누어 Average IO Request Interarrival Time (ioReqAvgIntArrTime)을 계산합니다. 즉 NIOREQ 개의 IO 요청이 totalCPUTime 기간 동안 발생하므로 IO 요청의 간격이 NIOREQ+1 개 존재한다고 가정합니다. 따라서 MIN_IOREQ_INT_ARRTIME 이 정해지면 최대 간격의 값은 (ioReqAvgIntArrTime - MIN_IOREQ_INT_ARRTIME)*2+ MIN_IOREQ_INT_ARRTIME 으로 결정됩니다.

CPU Scheduling Algorithms

다음의 CPU 스케쥴링 알고리즘을 구현합니다.

가) Round Robin Scheduling
Ready Queue에 들어오는 순서대로 CPU를 할당 받습니다. Quantum이 아주 크게 설정되면 (예 : INT_MAX) FCFS와 같은 결과를 갖게 됩니다.

나) Shortest Job First (Modified)
원래 Shortes Job First 는 Non-preemptive 스케줄링 방식이지만 quantum을 적용합니다. 즉 quantum이 끝나면 다시 Ready Queue로 보내지게 됩니다. 또는 IO Request를 한 경우에도 Blocked Queue에 있다 Ready Queue로 옮겨집니다. Ready Queue에 있는 프로세스 중 프로세스의 수행시간 (targetServiceTime) 이 제일 적은 프로세스가 다음에 CPU를 할당 받습니다.

다) Shortes Remaining Time Next (SRTN)
Ready Queue에서 현재 남아 있는 수행시간 (targetServiceTime - serviceTime) 이 가장 작은 프로세스를 선정하여 다음에 CPU를 할당합니다.

라) Guaranteed Scheduling (Modified)
초기에 설정되는 프로세스의 수행 시간(targetServiceTime)을 할당된 CPU 시간이라 가정하고 Ready Queue에 있는 프로세스 중에 serviceTime/targetServiceTime의 ratio가 가장 작은 프로세스가 다음에 CPU 할당 됩니다

마) Simple Feedback Scheduling
할당된 Quantum을 다 사용 못하고 IO Request 발생하는 프로세스를 IO bound 프로세스라 가정하고 우선순위(priority)를 증가 시킵니다 (io Done Event 로 중단된 경우는 우선순위를 변경하지 않습니다). 반대로 프로세스가 Quantum을 다 사용하고 강제적으로 스위치 되는 경우에는 CPU bound 프로세스라고 생각하고 priority를 감소합니다.
Ready Queue에 있은 프로세스 중에 priority가 가장 높은 값을 같는 프로세스를 다음에 CPU를 할당 합니다.

위의 모든 스케쥴링 알고리즘에서 선정기준이 되는 값이 동일한 경우의 프로세스들 사이에서는 FCFS 방식으로 선정합니다. Ready Queue에서 선정할 프로세스가 없는 경우에는 idleProc를 선정하여 CPU를 할당합니다.

4) 참고 및 주의 사항
schedulehw.c를 사용하여 프로그램을 완성합니다.

0) schedulehw.c를 사용하지만 첫 번째 주석 (학생 이름, 번호등)과 printResult(), compute() 함수, 제공되는 printf문 (인자이름은 변경 가능)을 제외한 모든 것을 바꾸어도 됩니다. main() 함수에서 printResult() 함수를 반드시 호출해야 합니다. (주의사항 : 제출 파일의 처음에 제출년도/과목명/과제명/학번/이름을 명시하시오)