5장: CPU 스케줄링

Silberschatz, Galvin 및 Gagne ©2018

5장: CPU 스케줄링

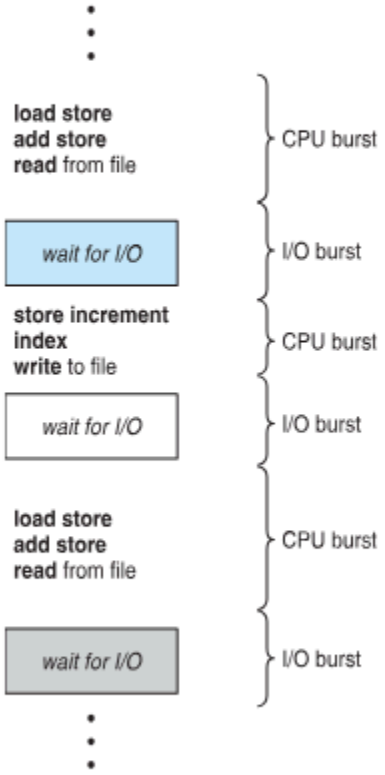
- 기본 개념스케줄링 기준스케
- 줄링 알고리즘스레드 스케줄
- 링다중 프로세서 스케줄링실
- 시간 CPU 스케줄링운영 체제
- 예제알고리즘 평가

목표

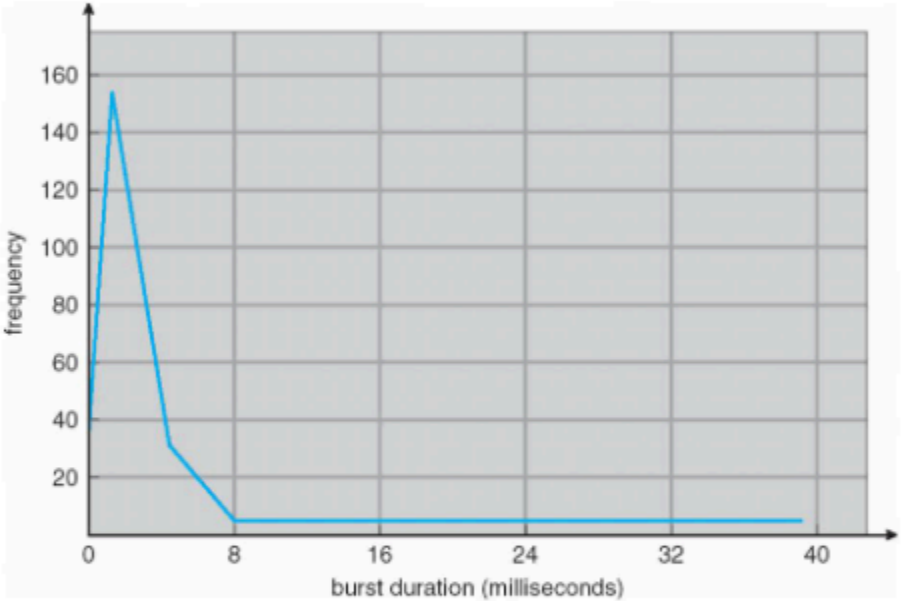
- 다중 프로그래밍된 운영 체제의 기초가 되는 CPU 스케줄링을 소개합니다.다양한 CPU 스케줄링 알고리즘을 설명하기 위해특정 시스템에 대한 CPU 스케줄링 알고
- 리즘을 선택하기 위한 평가 기준에 대해 논의하기 위해여러 운영 체제의 스케줄링
- 알고리즘을 검사하기 위해

기본 개념

- 다중 프로그래밍으로 얻을 수 있는 최대 CPU 사용률CPU–I/O 버스트 주기 – 프로
- 세스 실행은 CPU 실행 및 I/OwaitCPU 버스트 주기와 I/O 버스트CPU 버스트 분포로 구성됩니다.



CPU 버스트 시간의 히스토그램



CPU 스케줄러

- **단기 스케줄러는 준비 대기열에 있는 프로세스 중에서 선택하고 그 중 하나에 CPU를 할당합니다**

  대기열은 다양한 방법으로 정렬될 수 있습니다CPU 스케줄링 결정은 다음과 같은 경
- 우 발생할 수 있습니다.

  1. 실행 상태에서 대기 상태로 전환
  2. 실행 상태에서 준비 상태로 전환
  3. 대기 상태에서 준비로 전환

  4. 종료1 및 4 미만의 스케줄링은 비선점적임다른 모든 스케
- 줄링은 선점적임공유 데이터에 대한 액세스 고려커널 모드
- 중 선점 고려중요한 OS 활동 중에 발생하는 인터럽트 고려

발송자

- 디스패처 모듈은 단기 스케줄러가 선택한 프로세스에 CPU를 제어합니다. 여기에는 다음이 포함됩니다.

  컨텍스트 전환사용자 모드로 전환사용자 프로그램의 적절한 위치로 이동하여 해당 프로그램을 다시 시작디스패치 대기 시간 – 디스패처가 한 프로세스를 중지하고 다른 프로세스를 시작하는 데 걸리는 시간

-

스케줄링 기준

- CPU 사용률 – CPU를 가능한 한 바쁘게 유지처리량 – 시간당 실행을 완료하는
- 프로세스의 # 단위턴어라운드 시간 – 특정 프로세스를 실행하는 데 걸리는 시간
- 대기 시간 – 프로세스가 준비된 대기열에서 대기한 시간응답 시간 – 요청이 제출
- 된 시점부터 첫 번째 응답이 생성될 때까지 걸리는 시간, 출력되지 않음(시간 공
- 유 환경의 경우)

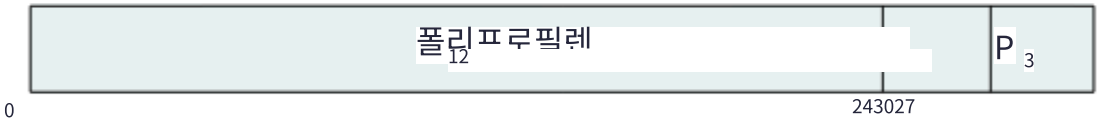
스케줄링 알고리즘 최적화 기준

- 최대 CPU 사용률최
- 대 처리량최소 처리
- 시간최소 대기 시간
- 최소 응답 시간
-

선착순(FCFS) 스케줄링

프로세스	버스트 시간
P1	24
P2	3
P3	3

- 프로세스가 P1 , P2 , P3 순서로 도착한다고 가정합니다.일정에 대한 Gantt 차트는 다음과 같습니다.



- P1 대기 시간 = 0; P2 = 24; P3 = 27평균 대기 시간: (0 + 24 + 27)/3 = 17
-

FCFS 스케줄링(계속)

프로세스가 다음 순서로 도착한다고 가정합니다.

P2 , P3 , P1

- 일정에 대한 Gantt 차트는 다음과 같습니다.

- P1 대기 시간 = 6; P2 = 0; P3 = 3평균 대기 시
- 간: (6 + 0 + 3)/3 = 3이전 사례보다 훨씬 우수호
- 송 효과 - 긴 프로세스 뒤에 짧은 프로세스
-
  - 하나의 CPU 바운드 프로세스와 여러 I/O 바운드 프로세스를 고려합니다.

SJF(Shortest-Job-First) 스케줄링

- 각 프로세스와 다음 CPU 버스트 길이를 연결합니다.

  이 길이를 사용하여 가장 짧은 시간으로 프로세스를 예약하십시오.SJF가 최적입니다
- – 주어진 프로세스 세트에 대한 최소 평균 대기 시간을 제공합니다.
  - 어려움은 다음 CPU 요청의 길이를 아는 것입니다.사용자에게 물어볼 수 있습니
  - 다.

SJF의 예

ProcessArriva l 시간		버스트 시간
P1	0.0	6
P2	2.0	8
P3	4.0	7
P4	5.0	3

- SJF 스케줄링 차트

- 평균 대기 시간 = (3 + 16 + 9 + 0) / 4 = 7

다음 CPU 버스트 길이 확인

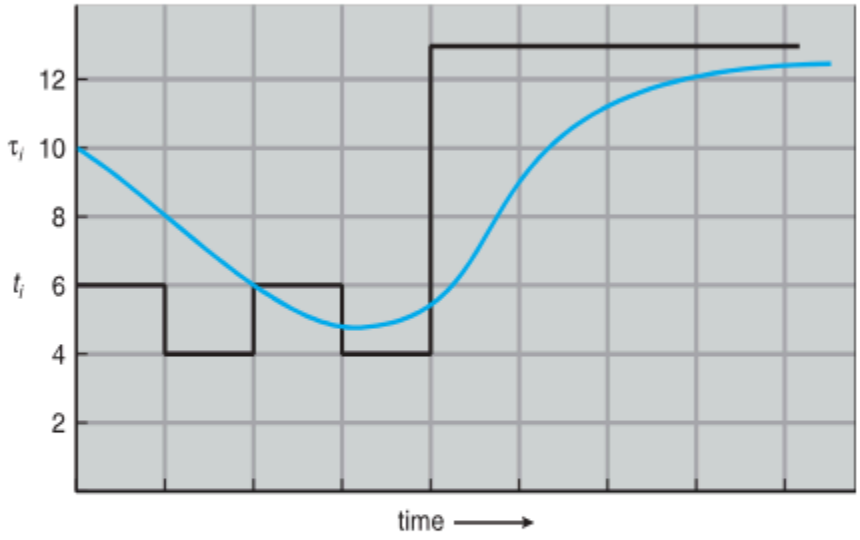
- 길이만 추정할 수 있습니다 – 이전 길이와 유사해야 합니다.
  - 그런 다음 가장 짧게 예측된 다음 CPU 버스트가 있는 프로세스를 선택합니다.

- 이전 CPU 버스트의 길이를 사용하여 지수 평균화를 사용하여 수행할 수 있습니다.

1. t_n = 전류 길이 의 n 일중앙 처리 유닛 파열하다
2. $\square_{n+1}$ = 예측 값 때문에 이 다음 중앙 처리 유닛 파열하다
3. $\square$ 0 $\square\square\square$ 1
4. 정의 : $\square_n == \square t + ()_1 - \square\square$

- 일반적으로 α shortest-remaining-time-first라고 하는 1/2Preemptive 버전으로
- 설정됩니다.

다음 CPU 버스트의 길이 예측



CPU burst (t_i)	6	4	6	4	13	13	13	...
"guess" (τ_i)	10	8	6	6	5	9	11	12	...

지수 평균화의 예

- □ = 0

  □n+1 = ◆(#9)◆n최근 기록은
  계산되지 않습니다□ =1

- □n+1 = □ tn실제 마지막 CPU 버스트만 계산됩니다.공식을 확장하면 다음과 같은 결과가 나타납니다.

  □n+1 = □ tn+(1 - □)□ tn -1 + ...
  +(1 - □)j □ tn -j +
  ...+(1 - □)n +1 □0

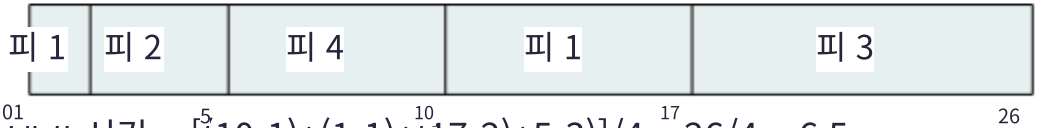
- □ 및 (1 - □)는 모두 1보다 작거나 같으므로 각 연속 항은 이전 항보다 가중치가 낮습니다

Shortest-lefting-time-first의 예

- 이제 다양한 도착 시간과 선점의 개념을 분석에 추가합니다

ProcessA	도착 도착 시간T	버스트 시간
P1	08	
P2	14	
P3	29	
P4	35	

- 선점형 SJF 간트 차트



- 평균 내기 시간 = [(10-1)+(1-1)+(17-2)+5-3)]/4 = 26/4 = 6.5msec

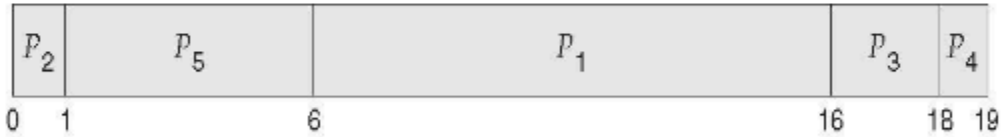
우선 순위 스케줄링

- 우선 순위 번호(정수)는 각 프로세스와 연결됩니다
- CPU는 가장 높은 우선 순위(가장 작은 정수 ☒가장 높은 우선 순위)를 가진 프로세스에 할당됩니다.
  - PreemptiveNonpreemptive
  -
- SJF는 우선 순위 스케줄링이며, 우선 순위는 예측된 다음 CPU 버스트 시간의 역수입니다
- 문제 ☒ 기아 – 우선 순위가 낮은 프로세스는 실행되지 않을 수 있습니다.
- 해결책 ☒ 노화 – 시간이 지남에 따라 프로세스의 우선 순위가 높아집니다.

우선 순위 스케줄링의 예

ProcessA	arri 버스트 TimeT	우선권
P1	10	3
P2	11	
P3	24	
P4	15	
P5	52	

- 우선 순위 스케줄링 간트 차트



- 평균 대기 시간 = 8.2msec

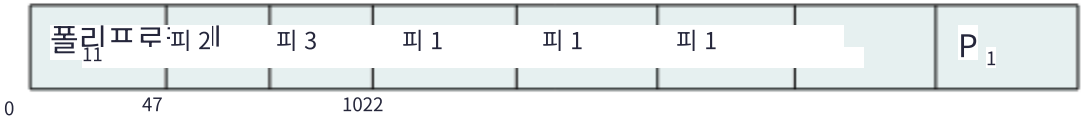
라운드 로빈 (RR)

- 각 프로세스는 일반적으로 10-100milliseconds의 작은 CPU 시간 단위(시간 양자 q)를 가져옵니다. 이 시간이 경과하면 프로세스가 선점되고 준비된 큐의 끝에 추가됩니다. 준비 대기열에 n개의 프로세스가 있고 시간 양자가 q인 경우 각 프로
- 세스는 한 번에 최대 q 시간 단위의 청크로 CPU 시간의 1/n을 가져옵니다. Noprocess는 (n-1)q 시간 단위보다 더 오래 기다립니다. 타이머는 다음 프로세스를 예약하기 위해 모든 퀀텀을 중단합니다.성능
-
-
  - q large ☒ FIFOq small ☒ q 컨텍스트 전환과 관련하여 커야 하며, 그렇지
  - 않으면 오버헤드가 너무 높습니다.

시간 양자 = 4를 사용하는 RR의 예

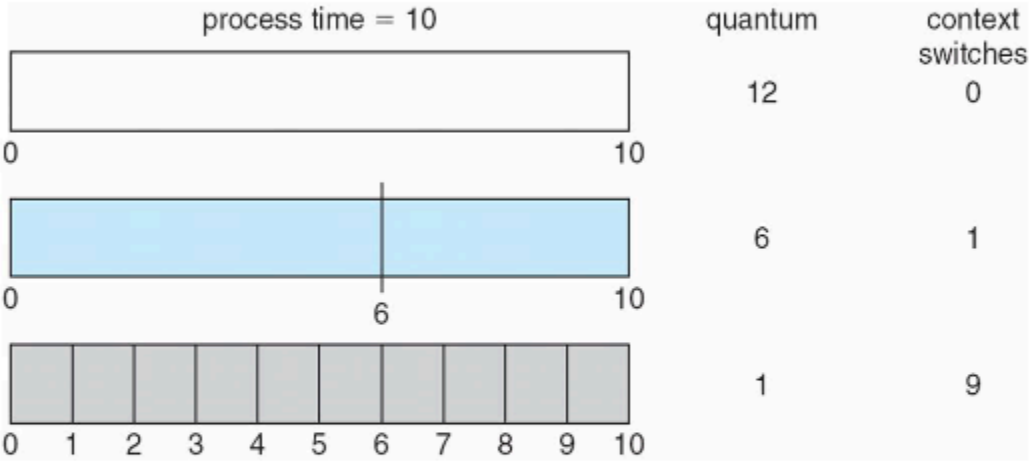
프로세스	버스트 시간
P1	24
P2	3
P3	3

- 간트 차트는 다음과 같습니다.

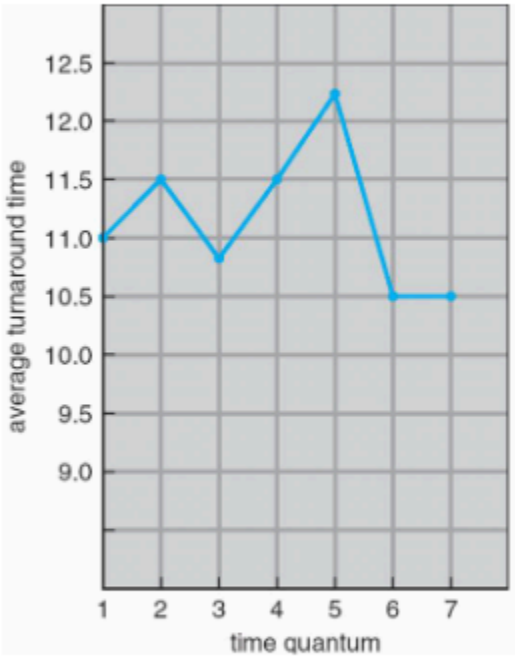


- 일반적으로 SJF보다 평균 턴어라운드가 높지만 더 나은 응답q는 컨텍스트 전환 시
- 간q에 비해 커야 합니다.q 일반적으로 10ms에서 100ms, 컨텍스트 전환 < 10usec
-

시간: 양자 및 컨텍스트, 시간 전환



처리 시간은 시간 퀀텀에 따라 다릅니다.



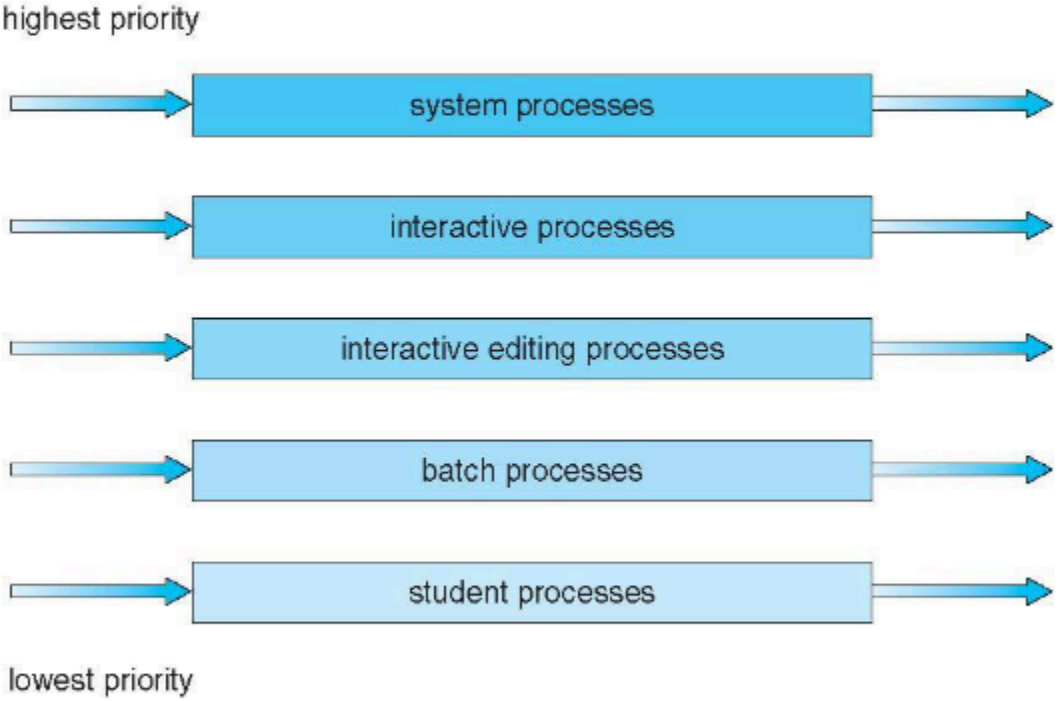
process	time
P_1	6
P_2	3
P_3	1
P_4	7

CPU 버스트의 80%는 q보다 짧아야 합니다.

다단계 대기열

- 준비 대기열은 별도의 대기열로 분할됩니다.
- **foreground (interactive)background (batch)주어진 queue에서 영구적으로 프**
- **로세스각 queue에는 자체 스케줄링 알고리**
- **즘이 있습니다.**
- foreground – RRbackground – FCFSScheduling은 대기열 간에 수행해야 합니다.
  - 고정 우선 순위 일정; (즉, 전경에서 모두 제공 한 다음 배경에서 제공). 굶주림의 가능성. 시간 조각 – 각 대기열은 프로세스 간에 예약할 수 있
  - 는 일정량의 CPU 시간을 가져옵니다. 즉, RR에서 전경으로 80%FCFS 에서 배경으로 20%
  -

다단계 대기열 스케줄링

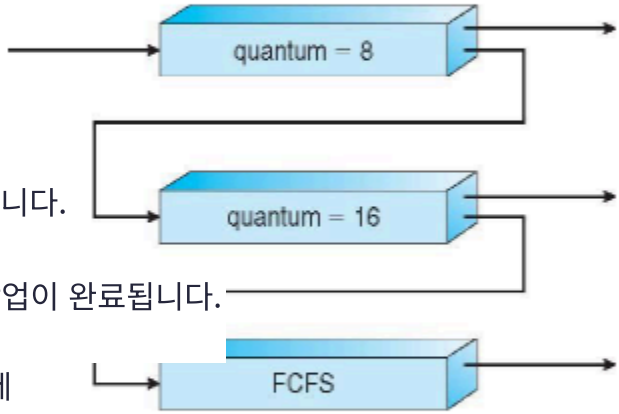

highest priority
system processes
interactive processes
interactive editing processes
batch processes
student processes
lowest priority

다단계 피드백 대기열

- 프로세스는 다양한 큐 간에 이동할 수 있습니다. 에이징은 다음 매개 변수로 정의된 Multilevel-feedback-queue 스케줄러를 통해 구현할 수 있습니다.
-
  - 큐의 수각 queue에 대한 스케줄링 알고리즘Process업그레이드 시기를 결
  - 정하는 데 사용되는 방법Process를 강등할 시기를 결정하는 데 사용되는
  - MethodProcess가 서비스를 필요로 할 때 프로세스가 들어갈 큐를 결정하
  - 는 데 사용되는 Method
  -

Multilevel Feedback Queue의 예

- 세 개의 대기열:
  - Q0 – 시간 양자가 8밀리초인 RRQ1 – RR
  - 시간 양자 16밀리초Q2 – FCFS
  -



- 일정
  - 새 작업이 FCFS로 제공되는 큐 Q0에 들어갑니다.

    ☒ CPU를 확보하면 작업은 8을 받습니다.
    밀리초☒ 8밀리초 내에 완료되지 않으면 작업이 완료됩니다.

  - Q1 대기열로 이동Q1 작업이 다시 FCFS에 제공되고 16추가 밀리초를 수신합니다.

    ☒ 그래도 완료되지 않으면 선점됩니다.
    Q2 대기열로 이동했습니다.

스레드 스케줄링

- 사용자 수준 스레드와 커널 수준 스레드의 차이점스레드가 지원되고 스레드가
- 예약되며 프로세스가 아님다대일 및 다대다 모델에서 스레드 라이브러리는 사
- 용자 수준 스레드가 LWPK에서 실행되도록 예약합니다.스케줄링 경쟁이 있기 때문에 PCS(process-contention scope)로 사용되지 않습니다.

프로세스 내일반적으로 프로그래머가 설정 한 우선 순위를 통해 수행됩니다.사용 가능한 CPU에 예약 된 커널 스레드는 시스템의 모든 스레드 간의 경쟁 인

- SCS (System-Contention Scope)입니다.

Pthread 스케줄링

- API를 사용하면 스레드 생성 중에 PCS 또는 SCS를 지정할 수 있습니다.

  PTHREAD_SCOPE_PROCESS는 PCS를 사용하여 스레드를 예약 schedulingPTHREAD_SCOPE_SYSTEM SCS 스케줄링을 사용하여 스레드를 예약
- 합니다.OS에 의해 제한될 수 있습니다(Linux 및 Mac OS X에만 해당 allowPTHREAD_SCOPE_SYSTEM

Pthread 스케줄링 API

```
#include <pthread.h>#include
<stdio.h>#define NUM_THREADS 5int
main(int argc, char *argv[]) {

    int i, 범위; pthread_t tid [스레드 수]; pthread_attr_t
    attr;/* 기본 속성 가져오기
    */pthread_attr_init(&attr);/* 현재 범위를 먼저 조회합니
    다. */if (pthread_attr_getscope(&attr, &scope) != 0)



    fprintf(stderr, "스케줄링 범위를 가져올 수 없습니다₩n"); 그렇지
    않으면 {
        if (범위 == PTHREAD_SCOPE_PROCESS)
        printf("PTHREAD_SCOPE_PROCESS"); 그렇지 않으면
        (범위 == PTHREAD_SCOPE_SYSTEM)
        printf("PTHREAD_SCOPE_SYSTEM"); 다른

    fprintf(stderr, "잘못된 범위 값.₩n");}
```

Pthread 스케줄링 API

```
/* 스케줄링 알고리즘을 PCS 또는 SCS로 설정
*/pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM);/* 스레
드 생성 */for (i = 0; i < NUM_THREADS; i++)


pthread_create(&tid[i],&attr,runner,NULL);/* 이제 각
스레드에서 조인 */for (i = 0; i < NUM_THREADS; i++)

pthread_join(tid[i], NULL);}/* 각 스레드는 이 함수에서 제어를
시작합니다 */void *runner(void *param){



/* 작업 좀 해봐 ...
*/pthread_exit(0);}
```

다중 프로세서 스케줄링

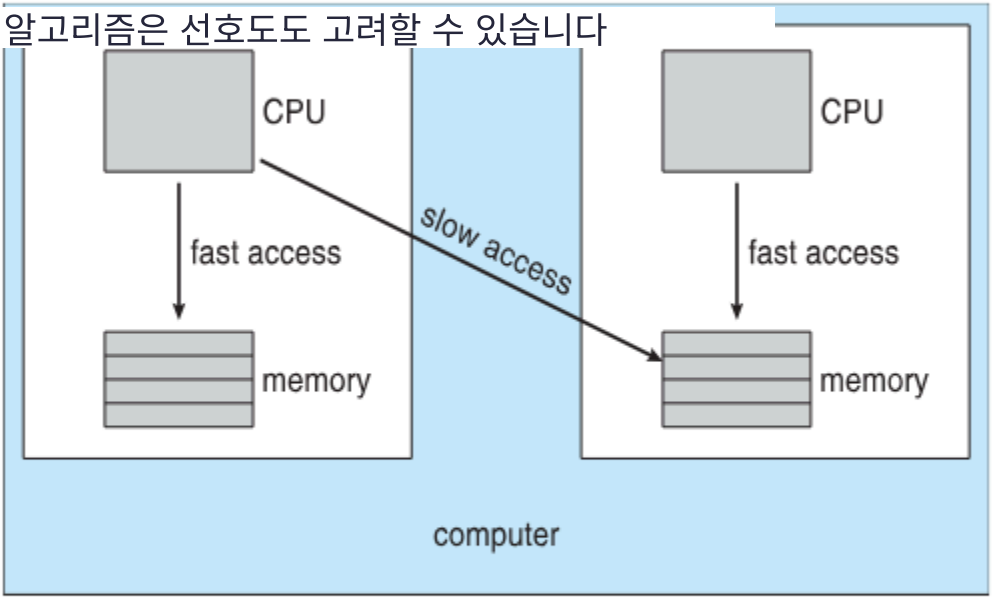
- 여러 CPU를 사용할 수 있는 경우 CPU 스케줄링이 더 복잡함다중 프로세서 내의
- 동종 프로세서비대칭 다중 처리 – 하나의 프로세서만 시스템 데이터 구조에 액세
- 스하여 데이터 공유의 필요성을 완화합니다.대칭 다중 처리(SMP) – 각 프로세서
- 는 자체 스케줄링, 모든 프로세스가 공통 준비 대기열에 있거나 각 프로세서에 준
- 비된 프로세스의 고유한 개인 대기열이 있습니다.

- 현재 가장 일반적인Processor 선호도 – 프로세스는 현재 실행중인 프로세서에
- 대한 선호도를 가지고 있습니다.

  - **soft affinityhard affinity프로**
  - **세서 세트를 포함한 변형**
  -

NUMA 및 CPU 스케줄링

메모리 배치 알고리즘은 선호도도 고려할 수 있습니다



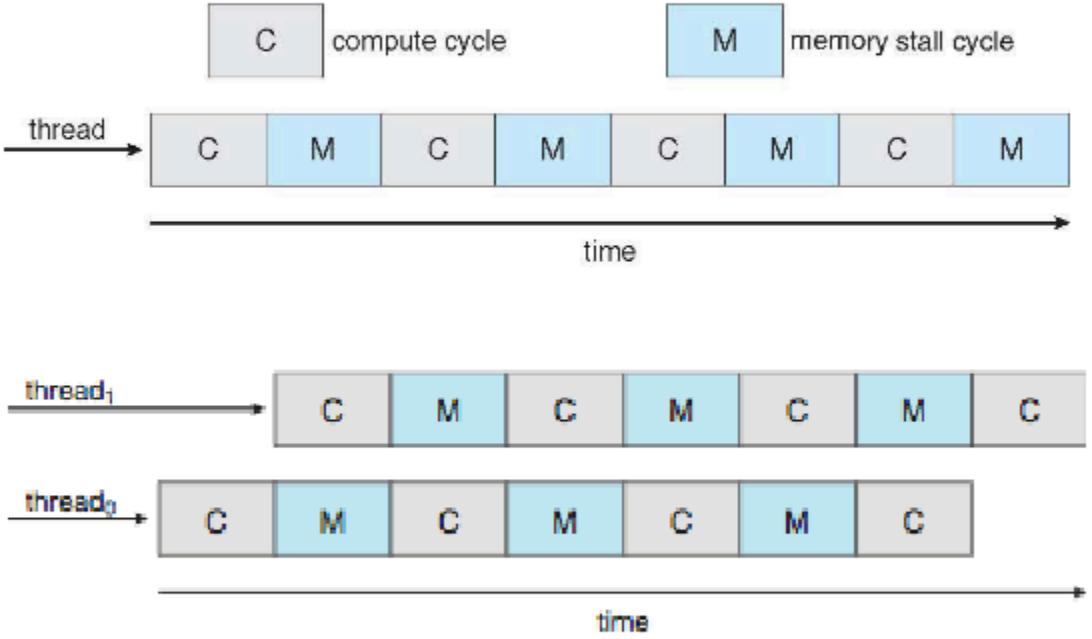
Multiple-Processor Scheduling – 로드 밸런싱

- SMP인 경우 효율성을 위해 모든 CPU를 로드된 상태로 유지해야 합니다
- 로드 밸런싱을 통해 워크로드를 균등하게 분산하려고 시도합니다.마이그
- 레이션 푸시 – 각 프로세서에 대한 부하를 주기적으로 확인하고, 발견된 경우 오버로드된 CPU에서 다른 CPU로 작업을 푸시마이그레이션 – 유휴 프
- 로세서가 사용 중인 프로세서에서 대기 중인 작업을 가져옵니다.

멀티코어 프로세서

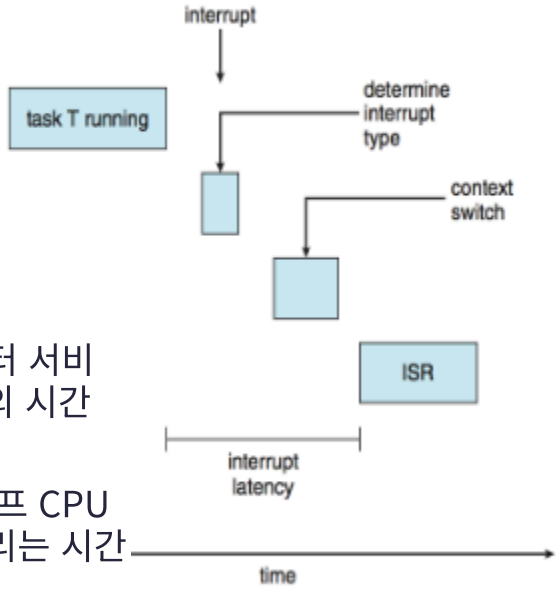
- 동일한 물리적 칩에 여러 프로세서 코어를 배치하는 최근 추세더 빠
- 르고 더 적은 전력을 소비합니다코어당 다중 스레드도 증가하고 있
- 습니다.
  - 메모리 검색이 발생하는 동안 다른 스레드에서 진행하기 위해 메모리 지연을 활용합니다.

멀티스레드 멀티코어 시스템



실시간 CPU 스케줄링

- 명백한 문제를 제시할 수 있음소프트
- 실시간 시스템 – 중요한 실시간 프로세스가 언제 예약될지 보장할 수 없음
  하드 실시간 시스템 – 작업은 기한까지 처리되어야 함두 가지 유형의 지연 시간이 성능에 영향을 미칩니다.
-

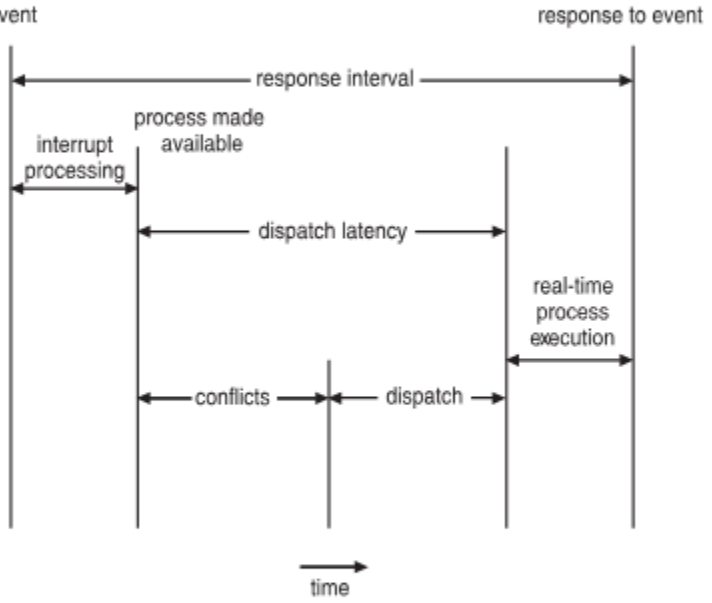


1. 인터럽트 대기 시간 – 인터럽트 도착부터 서비스가 인터럽트하는 루틴이 시작될 때까지의 시간

2. 디스패치 대기 시간 – 현재 프로세스 오프 CPU를 가져 와서 다른 CPU로 전환하는 데 걸리는 시간

실시간 CPU 스케줄링(계속)

- dispatchlatency의 충돌 단계:
  1. inkernel 모드를 실행하는 모든 프로세스의 선점
  2. high-priorityprocesses에 의해 필요한 자원의 low-priority 프로세스에 의한 릴리스

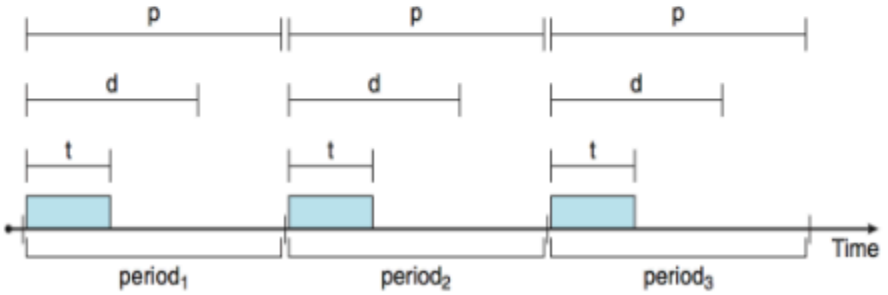


우선 순위 기반 스케줄링

- 실시간 스케줄링의 경우 스케줄러는 선점형 우선 순위 기반 스케줄링을 지원해야 합니다

  그러나 소프트 실시간만 보장합니다.하드 실시간의 경우 마감 시간을 충
- 족할 수 있는 기능도 제공해야 합니다.프로세스에는 새로운 특성이 있습
- 니다: 주기적인 프로세스에는 일정한 간격으로 CPU가 필요합니다.

  - 처리 시간 t, 기한 d, 기간 p0 ≤ t ≤ d
  - ≤ p주기적 작업의 비율은 1/p입니다.
  -



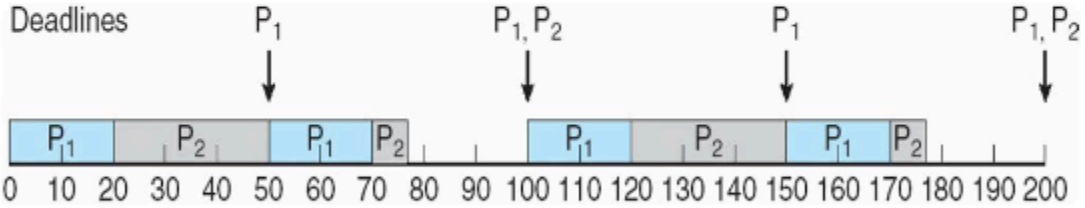
가상화 및 스케줄링

- 가상화 소프트웨어는 여러 게스트를 CPU에 스케줄링합니다.각 게스트는 자체 스케줄
- 링을 수행합니다.

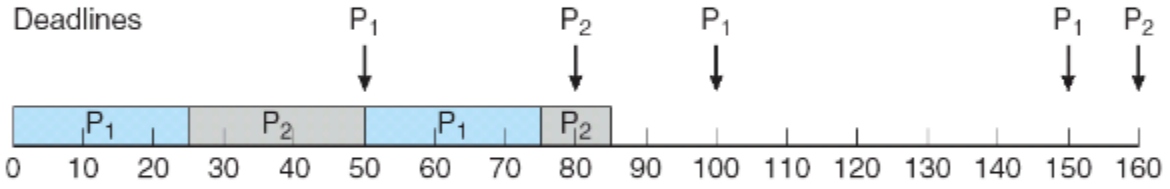
  CPU를 소유하고 있지 않다는 것을 모르면 응답 시간
  이 느려질 수 있습니다.게스트의 시간 시계에 영향을
  줄 수 있습니다게스트의 좋은 스케줄링 알고리즘 노
- 력을 취소할 수 있습니다.

Montonic 스케줄링 평가

- 우선 순위는 기간의 역순을 기준으로 할당됩니다
- 더 짧은 기간 = 더 높은 우선 순위;
- 더 긴 기간 = 낮은 우선 순위
- P1에는 P2보다 높은 우선 순위가 할당됩니다.



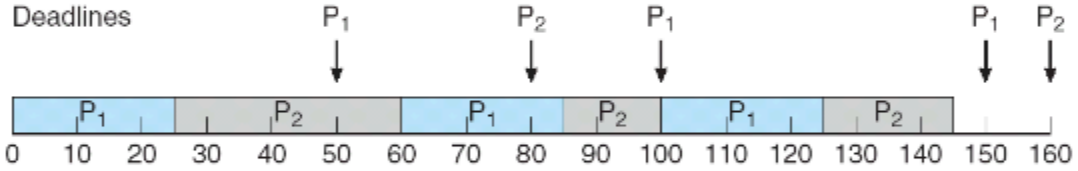
Rate Monotonic Scheduling으로 마감 시간 놓침



가장 빠른 마감일 첫 번째 스케줄링(EDF)

- 우선 순위는 마감일에 따라 할당됩니다.

  마감일이 빠를수록 우선 순위가 높아집니다. 마감 시한이 늦을수록 우선 순위가 낮아집니다.



비례 주식 스케줄링

- T 공유는 시스템의 모든 프로세스에 할당됩니다
- 응용 프로그램은 N이 T인 N개의 공유를 수신<
- 이렇게 하면 각 애플리케이션이 총 프로세서 시간의 N/T를 수신하게 됩니다

POSIX 실시간 스케줄링

- POSIX.1b 표준
- API는 실시간 스레드를 관리하기 위한 기능을 제공합니다.
- 실시간 스레드에 대한 두 개의 스케줄링 클래스를 정의합니다.

1. SCHED_FIFO - 스레드는 FIFO queue와 함께 FCFS 전략을 사용하여 예약됩니다. 우선 순위가 같은 스레드에 대한 시간 분할은 없습니다

2. SCHED_RR - 우선 순위가 동일한 스레드에 대해 시간 슬라이싱이 발생한다는 점을 제외하고는 SCHED_FIFO과 유사합니다.

- 스케줄링 정책을 가져오고 설정하기 위한 두 가지 기능을 정의합니다.

1. `pthread_attr_getsched_policy(pthread_attr_t *attr, int *정책)`
2. `pthread_attr_setsched_policy(pthread_attr_t *attr, int 정책)`

POSIX 실시간 스케줄링 API

```
#include <pthread.h>#include
<stdio.h>#define NUM_THREADS 5int
main(int argc, char *argv[]){

    int i, 정책; pthread_t_tid[NUM_THREADS]; pthread_attr_t
    attr;/* 기본 속성 가져오기 */pthread_attr_init(&attr);/* 현
    재 스케줄링 정책 가져오기 */if
    (pthread_attr_getschedpolicy(&attr, &policy) != 0)


    fprintf(stderr, "정책을 가져올 수 없습니다.₩n"); 그
    렇지 않으면 {
    if (정책 == SCHED_OTHER) printf("SCHED_OTHER₩n"); 그렇지 않으면
    (정책 == SCHED_RR) printf("SCHED_RR₩n"); 그렇지 않으면 (policy
    == SCHED_FIFO) printf("SCHED_FIFO₩n");}
```

POSIX 실시간 스케줄링 API(계속)

```
    /* 스케줄링 정책 설정 - FIFO, RR 또는 OTHER */if
    (pthread_attr_setschedpolicy(&attr, SCHED_FIFO) != 0)
    fprintf(stderr, "정책을 설정할 수 없습니다.₩n");/*
    스레드 생성 */for (i = 0; i < NUM_THREADS; i++)


    pthread_create(&tid[i],&attr,runner,NULL);/* 이제
    각 스레드에서 조인 */for (i = 0; i < NUM_THREADS;
    i++)
pthread_join(시간[i], 제로);}



/* 각 스레드는 이 함수에서 제어를 시작합니다 */void
*runner(void *param){


/* 작업 좀 해봐 ...
*/pthread_exit(0);}
```

운영 체제 예제

- Linux 스케줄링
- Windows 일정 관리
- Solaris 스케줄링

버전 2.5를 통한 Linux 스케줄링

- 커널 버전 2.5 이전에는 표준 UNIX 스케줄링 알고리즘의 변형버전 2.5가 일정 순서
- O(1) 스케줄링 시간으로 이동했습니다.
  - 선점, 우선 순위 기반두 가지 우선 순위 범위: 시간 공유 및 실시간0에서
  - 99까지의 실시간 범위와 100에서 140까지의 nice 값더 높은 우선 순위
  - 를 나타내는 숫자로 더 낮은 값으로 전역 우선 순위에 매핑더 높은 우선
  - 순위는 시간 조각에 남은 시간만큼 더 큰 qTask 실행 가능(활성)시간이 남아 있지 않은 경우(만료됨) 다른 모든 작업이 해당 조각을 사용할 때까
  - 지 실행할 수 없음CPU당 실행 대기열 데이터 구조에서 추적되는 모든
  - 실행 가능한 작업
  -
  -

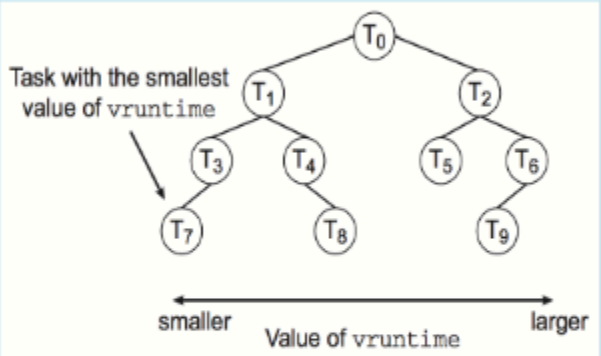
    ☒ 두 개의 우선 순위 배열(활성, 만료됨)☒ 우선 순위에 의해 인덱싱된 작업☒ 더 이상 활성 상태가 아닐 때 배열이 교환됩니다.잘 작동했지만 대화형 프로세스에 대한 응답 시간이 좋지 않습니다.
  -

버전 2.6.23의 Linux 스케줄링 +

- CFS(Completely Fair Scheduler)스케줄링 클래스
-
  - 각각은 특정 우선 순위를 갖습니다스케줄러는 고정 시간 할당을 기반으로 하는 양자가
  - 아니라 포함된 CPU 시간2 스케줄링 클래스의 비율에 따라 다른 작업을 추가할 수 있습
  - 니다.
  -
    1. 기본값
    2. 실시간
- 양자는 -20에서 +19 사이의 nice 값을 기반으로 계산됩니다.
  - 값이 낮을수록 우선 순위가 높습니다.목표 대기 시간 계산 – 작업이 한 번 이상 실행되
  - 어야 하는 시간 간격활성 작업 수가 증가하면 대상 대기 시간이 증가할 수 있습니다.
  -
- CFS 스케줄러는 vruntime 변수에서 작업별 가상 런타임을 유지 관리합니다.
  - 작업의 우선 순위에 따라 감쇠 계수와 관련 - 우선 순위가 낮을수록 감쇠율이 높음일반 기본
  - 우선 순위는 가상 실행 시간 = 실제 실행 시간을 산출합니다.
- 실행할 다음 작업을 결정하기 위해 스케줄러는 가상 실행 시간이 가장 낮은 작업을 선택합니다.

CFS 성능

The Linux CFS scheduler provides an efficient algorithm for selecting which task to run next. Each runnable task is placed in a red-black tree—a balanced binary search tree whose key is based on the value of `vruntime`. This tree is shown below:

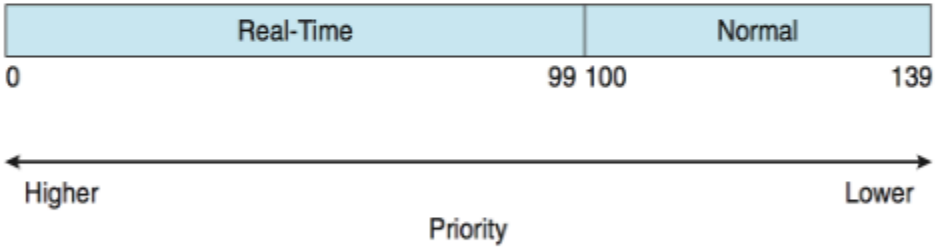


When a task becomes runnable, it is added to the tree. If a task on the tree is not runnable (for example, if it is blocked while waiting for I/O), it is removed. Generally speaking, tasks that have been given less processing time (smaller values of `vruntime`) are toward the left side of the tree, and tasks that have been given more processing time are on the right side. According to the properties of a binary search tree, the leftmost node has the smallest key value, which for the sake of the CFS scheduler means that it is the task with the highest priority. Because the red-black tree is balanced, navigating it to discover the leftmost node will require $O(\lg N)$ operations (where N is the number of nodes in the tree). However, for efficiency reasons, the Linux scheduler caches this value in the variable `rb_leftmost`, and thus determining which task to run next requires only retrieving the cached value.

Linux 스케줄링(계속)

- POSIX.1b에 따른 실시간 스케줄링

  실시간 작업에는 정적 우선 순위가 있음실시간 플러스
- 노멀 맵은 전역 우선 순위 체계에 매핑Nice 값 -20은
- 전역 우선 순위 100에 매핑니스 값 +19는 우선 순위
- 139에 매핑됩니다.



Windows 스케줄링

- Windows는 우선 순위 기반 선점형 스케줄링을 사용합니다.가장 높은 우선 순위의 스레드 실행 nextDispatcher는 스케줄러입니다.스레드는 (1) 블록, (2) 시간 조각을 사용, (3) 더 높은 우선 순위에 의해 선점될 때까지 실행됩니다.스레드실시간 스레드는 비실시간을 선점할 수 있습니다.32 수준 우선 순위 schemeVariable 클래스는 1-15, 실시간 클래스는 16-31입니다.우선 순위 0 은 메모리 관리 스레드입니다.각 우선 순위에 대한 대기열실행 가능한 스레드가 없으면 유휴 스레드를 실행합니다.

Windows 우선 순위 클래스

- Win32 API는 프로세스가 속할 수 있는 몇 가지 우선 순위 클래스를 식별합니다

  REALTIME_PRIORITY_CLASS, HIGH_PRIORITY_CLASS,ABOVE_NORMAL_PRIORITY_CLASS,NORMAL_PRIORITY_CLASS,BELOW_NORMAL_PRIORITY_CLASS IDLE_PRIORITY_CLASSAll는 변수이지만 지정된 우선 순위 클래스 내의 REALTIMEA 스레드는 상대적 우선 순위를 갖습니다.

- TIME_CRITICAL, HIGHEST, ABOVE_NORMAL, NORMAL,BELOW_NORMAL, LOWEST, IDLEPriority 클래스 및 상대 우선 순위가 결합되어 숫자 우선 순위를 제공합니다.기본 우선 순위는 클래스 내에서 NORMAL입니다.양자가 만료되면 우선 순위가 낮아지지만 기본 이하로는 내려가지 않습니다.

Windows 우선 순위 클래스(계속)

- 대기가 발생하면 대기한 항목에 따라 우선 순위가 높아짐3배 우
- 선 순위가 지정된 포그라운드 창Windows 7에 UMS(사용자 모
- 드 스케줄링)가 추가되었습니다.
  - 응용 프로그램은 커널과 독립적으로 스레드를 생성하고 관리합니다.스레드 수가
  - 많을수록 훨씬 더 효율적입니다UMS 스케줄러는 C++ ConcRT(Concurrent
  - Runtime)와 같은 프로그래밍 언어 라이브러리에서 제공됩니다.

Windows 우선 순위

	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

솔라리스

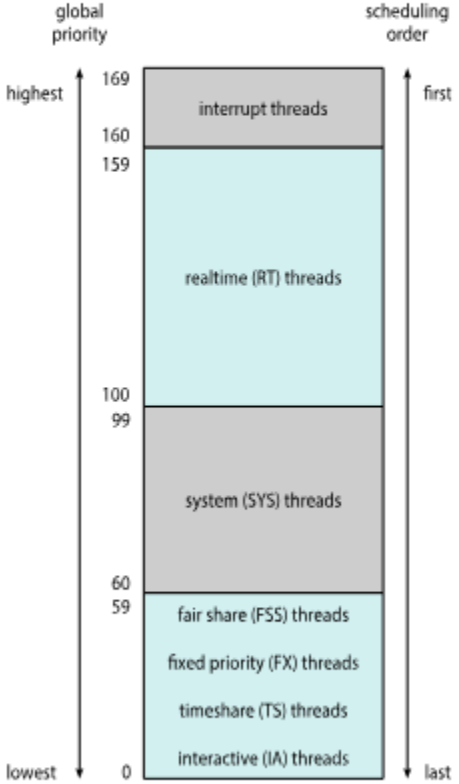
- 우선 순위 기반 스케줄링6개의 클래스 사용 가능
-

  시간 공유(기본값) (TS)인터랙티브 (IA)실
  시간 (RT)시스템 (SYS)공정 분배 (FSS)고
  정 우선 순위 (FP)주어진 스레드는 한 번에
  하나의 클래스에 있을 수 있습니다각 클래
  스에는 자체 스케줄링 알고리즘이 있습니
  다시간 공유는 다중 레벨 피드백 대기열입
- 니다.
-
-
  - sysadmin에서 구성할 수 있는 로드 가능한 테이블

Solaris 디스패치 테이블

priority	time quantum	time quantum expired	return from sleep
0	200	0	50
5	200	0	50
10	160	0	51
15	160	5	51
20	120	10	52
25	120	15	52
30	80	20	53
35	80	25	54
40	40	30	55
45	40	35	56
50	40	40	58
55	40	45	58
59	20	49	59

Solaris 스케줄링



Solaris 스케줄링(계속)

- 스케줄러는 클래스별 우선 순위를 스레드별 전역 우선 순위로 변환합니다
  - 우선 순위가 가장 높은 스레드는 (1) 블록, (2) 시간 조각 사용, (3) 더 높은 우선 순위에 의해 선점될 때까지 실행threadRR을 통해 선택된 동일한 우선 순위의 여러 스레드 실행
  -

알고리즘 평가

- OS에 대한 CPU 스케줄링 알고리즘을 선택하는 방
- 법은 무엇입니까? 기준을 결정한 후 알고리즘 평가
- 결정론적 모델링

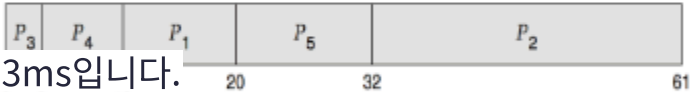
  분석 평가 유형미리 결정된 특정 작업 부하를 사용하여 해당 작업 부하에 대한 각 알고리즘의 성능을 정의합니다.시간 0에 도착하는 5개의 프로세스를 고려합니다.

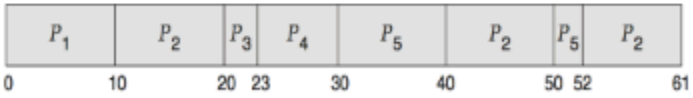
Process	Burst Time
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

결정론적 평가

- 각 알고리즘에 대해 최소 평균 대기 시간을 계산합니다.간단하고 빠르지만 입력에 정
- 확한 숫자가 필요하며 해당 입력에만 적용됩니다.
  - FCS는 28ms입니다.
  - 비선점 SFJ는 13ms입니다.

  - RR은 23ms입니다.





모델 대기열 처리

- 프로세스의 도착, CPU 및 I/O 버스트를 확률적으로 설명합니다.

  일반적으로 기하급수적이며 mean으로 설명됩니다.평균 처리량, 활용도, 대기 시간 등을 계산합니다.컴퓨터 시스템은 각각 대기 대기열이 있는 서버 네
- 트워크로 설명됩니다.프로세스.

  - 도착 속도 및 서비스 요금 파악활용도, 평균 대기열 길이, 평균 대기 시간 등을 계
  - 산합니다.

리틀의 포뮬러

- n = 평균 대기열 길이W = 대기열의 평균 대기 시간λ = queueLittle의 법
- 칙에 대한 평균 도착률 – 정상 상태에서 대기열을 떠나는 프로세스는
- processesarriving과 같아야 합니다.
-

$$n = \lambda \times W$$

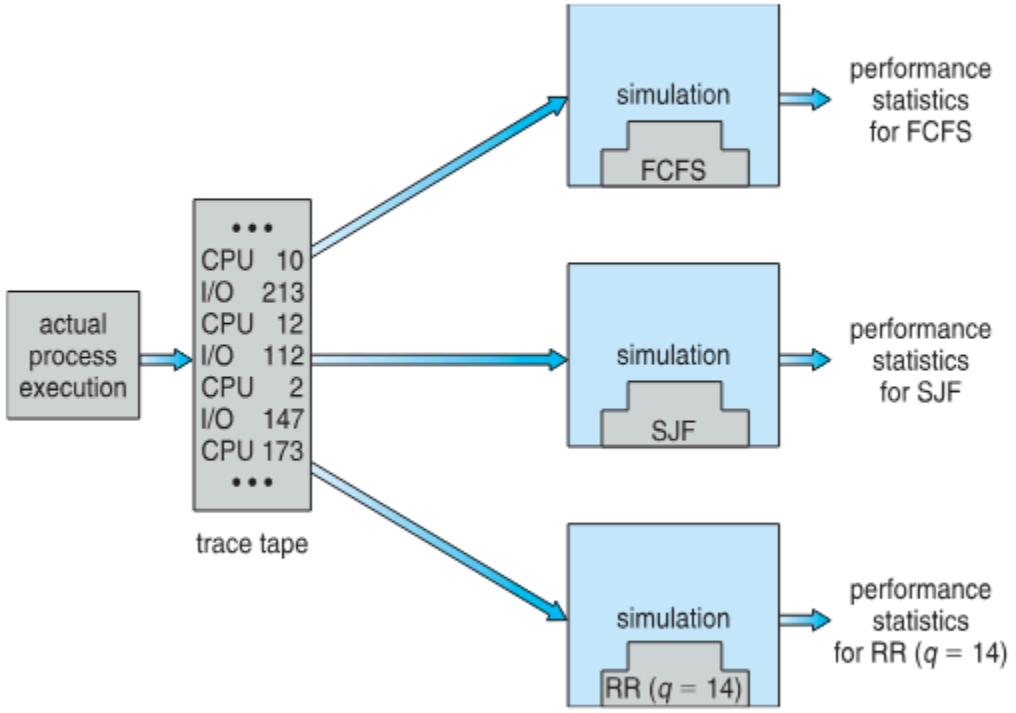
- 모든 스케줄링 알고리즘 및 도착 분포에 유효예를 들어, 평균적으로 초당 7개의 프로세스가 도착하고 일반적으로 14개의 프로세스가 대기열에 있는 경우 프로세스당 평균 대기 시간 = 2초

시뮬레이션

- 대기열 모델 제한시뮬레이션의 정확도 향상
-
  - 컴퓨터 시스템의 프로그래밍된 모델Clock은 알고
  - 리즘 성능을 나타내는 변수Gather 통계량입니다.
  - 데이터를 통해 수집된 시뮬레이션을 구동하기 위
  - 한 데이터

    ☒ 확률에 따른 난수 생성기☒ 수학적 또는 경험적으로 정의된 분포☒ 추적 테이프는 실제 시스템에서 실제 이벤트의 시퀀스를 기록합니다.

시뮬레이션을 통한 CPU 스케줄러 평가



이행

- 시뮬레이션조차도 정확도가 제한되어 있습니다.새로운 스케줄러를 구현하고 실제 시스템에서 테스트하기만 하면 됩니다.
- 높은 비용, 높은 위험환경이 다양함대부분의 유연한 스케줄러는 사이트별 또는 시스템별로 수정할 수 있습니다.또는 API를 수정하여 우선 순위를 수정할 수 있습니다.그러나 다시 환경은 다양합니다.

챕터 5의 끝

Silberschatz, Galvin 및 Gagne ©2018