Operating System Principles

操作系统原理

Process/Thread Scheduling

李旭东

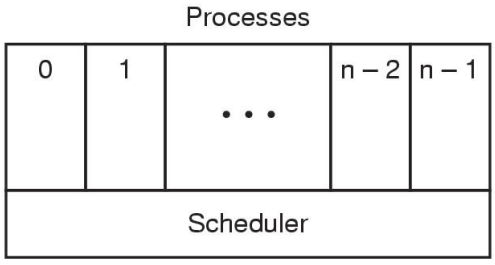
leexudong@nankai.edu.cn
Nankai University

Objectives

- Scheduler
- Process Behavior
- Scheduling Mode
- Scheduling Criteria
- Scheduling Algorithms
- Thread Scheduling

MultiProgramming

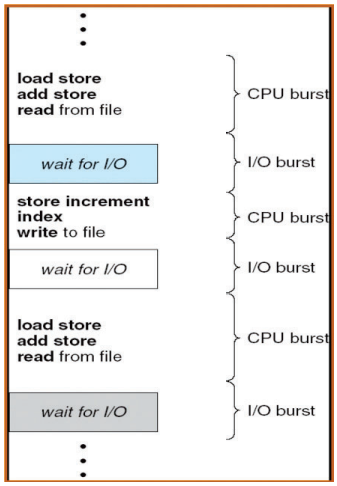
- Scheduler
- Scheduling algorithm



Scheduler

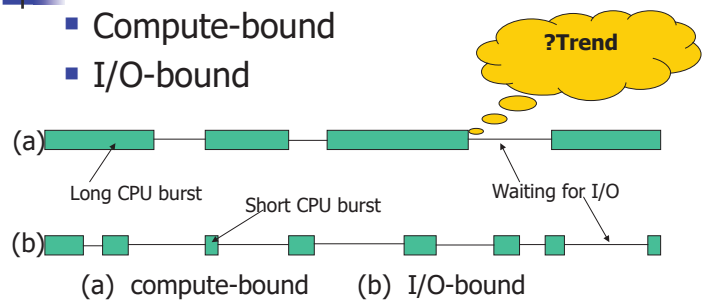
- Short-term Scheduler
  - CPU
- Middle-term Scheduler
  - Memory
- Long-term Scheduler
  - Job

CPU And I/O Bursts in a Process

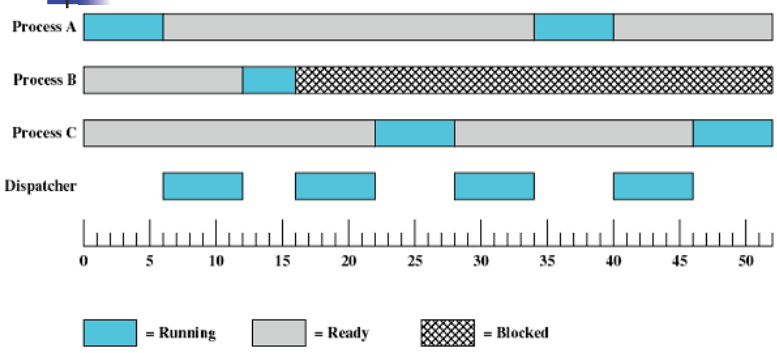


Process Behavior

- Compute-bound
- I/O-bound



Multi-Processes Trace



When to Schedule

- A new process is created
- A process exits
- A process blocks on I/O, on a semaphore, or for some other reason
- An I/O interrupt occurs

Dispatcher

- A module that gives control of the CPU to the process selected by the short-term scheduler
  - Switching context
  - Switching to user mode
  - Jumping to the proper location in the user program to restart that program
- Dispatch latency 调度延迟
  - The time it takes for the dispatcher to stop one process and start another running

Scheduling Modes

- Preemptive
  - 抢占式
- Nonpreemptive
  - 非抢占式，非剥夺式

Categories of Scheduling Algorithms

- Batch
- Interactive
- Real-time

Scheduling Criteria

- CPU utilization 利用率
- Throughout 吞吐量
- Turnaround time 周转时间
  - Waiting to get into memory
  - Waiting in the ready queue
  - Executing on the CPU
  - Doing I/O
- Waiting time
- Response time
- ...

Scheduling Algorithm Goals

All systems

Fairness - giving each process a fair share of the CPU
Policy enforcement - seeing that stated policy is carried out
Balance - keeping all parts of the system busy

Batch systems

Throughput - maximize jobs per hour
Turnaround time - minimize time between submission and termination
CPU utilization - keep the CPU busy all the time

Interactive systems

Response time - respond to requests quickly
Proportionality - meet users' expectations

Real-time systems

Meeting deadlines - avoid losing data
Predictability - avoid quality degradation in multimedia systems

Scheduling in Batch System

- First-come first-served
- Shortest job first
- Shortest remaining Time next

First-come first-served

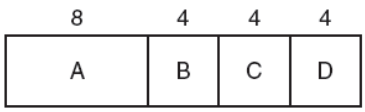
- Average waiting time

Process	Burst Time
P_1	24
P_2	3
P_3	3

Awt=(0+24+27)/3=17

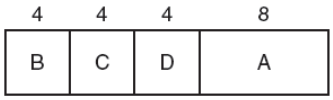
Awt=?

Shortest job first



(a)

1.nonpreemptive
2.preemptive



(b)

Figure 2-40. An example of shortest job first scheduling.
(a) Running four jobs in the original order.
(b) Running them in shortest job first order.

Quiz

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

SJF: AWT=?

FCFS: AWT=?

Shortest job first

- How to predict length of the next CPU burst?
  - exponential average

$T_{n+1}=at_n+(1-a)T_{n-1},\ 0\leqslant a\leqslant 1$
t_n *the length of the nth CPU burst*
T_{n+1} *the predicted value of the next CPU burst*

$$T_{n+1}=at_n+(1-a)at_{n-1}+\cdots+(1-a)^j at_{n-j}+\cdots+(1-a)^{n+1}T_0$$

Shortest remaining Time next

- i.e. Preemptive SJF sheduling

Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

Nonpreemptive SJF scheduling: AWT=?

Preemptive SJF scheduling: AWT=?

Scheduling in Interactive System

- Round-Robin Scheduling
- Priority Scheduling
- Multiple Queues
- Shortest Process Next
- Guaranteed Scheduling
- Lottery Scheduling
- Fair-Share Scheduling

Round-Robin Scheduling

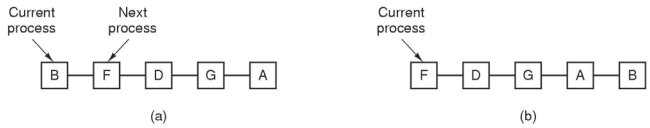


Figure 2-41. Round-robin scheduling.
(a) The list of runnable processes.
(b) The list of runnable processes after B uses up its quantum.

Round-Robin Scheduling

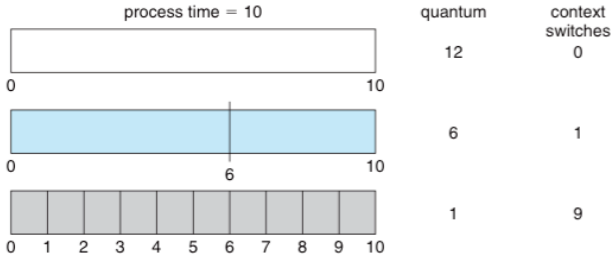
Process	Burst Time
P_1	24
P_2	3
P_3	3

a time quantum of 4 milliseconds

AWT=(6+4+7)/3=5.66

Quantum Value

- How a smaller time quantum increases context switches



Priority Scheduling

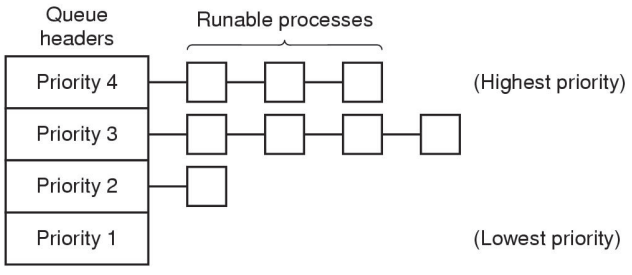
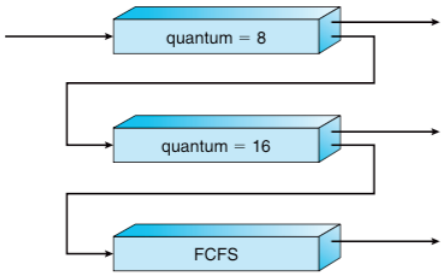


Figure. A scheduling algorithm with four priority classes.

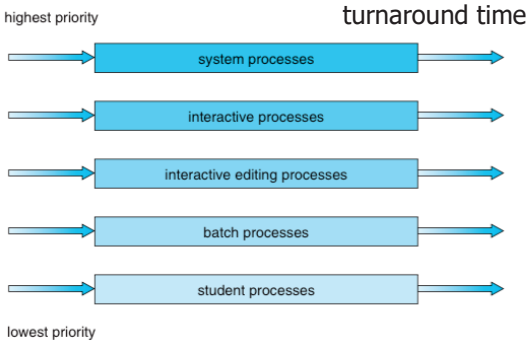
Mutlilevel Feedback Queue Scheduling

- Idea
  - Separate processes with different CPU-burst characteristics
  - Allow a process to move between queues



Multilevel Queue Scheduling

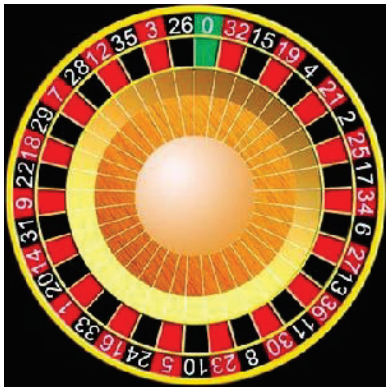
- Foreground (interactive) processes
- Background (batch) processes



Mutli-level Feedback Queue Scheduling

- the scheduler is defined by the following parameters:
  - The number of queues
  - The scheduling algorithm for each queue
  - The method used to determine when to upgrade a process to a higher-priority queue
  - The method used to determine when to demote a process to a lower-priority queue
  - The method used to determine which queue a process will enter when that process needs service

Lottery Scheduling



More Scheduling Algorithms

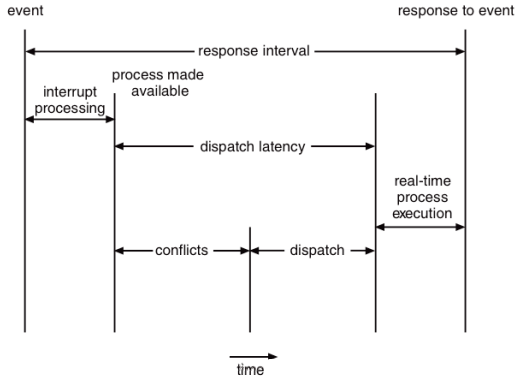
- Guaranteed Scheduling
- Fair-Share Scheduling
- ...

Scheduling in Real-time System

- Categories I
  - Hard real time
  - Soft real time
- Categories II
  - Periodic
  - Aperiodic
- Categories III
  - Static
  - dynamic

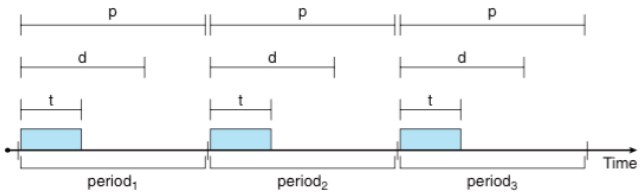
Real-time CPU Scheduling

- Minimizing Latency 最小化延迟
  - Interrupt latency, Dispatch latency

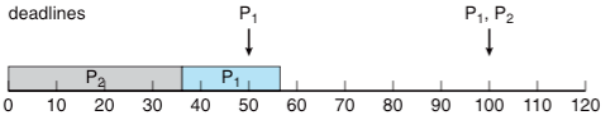


Real-time CPU Scheduling

- Priority-Based Scheduling



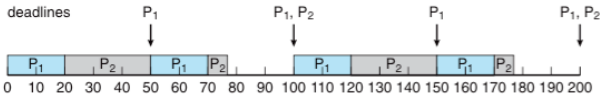
Example: P2 has a higher priority than P1



Real-time CPU Scheduling

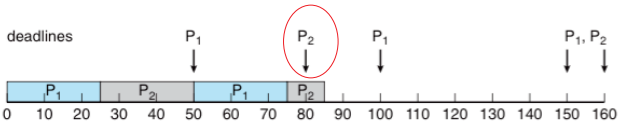
- Rate-Monotonic Scheduling
  - 单一速率
  - a static priority policy with preemption

Example:
P1 a higher priority than P2;
the period of P1 is shorter than that of P2

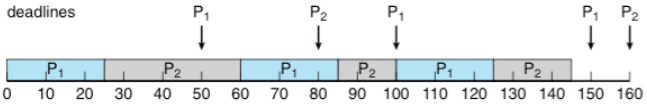


Real-time CPU Scheduling

- Earliest-Deadline-First Scheduling
  - dynamically assigns priorities according to deadline



Missing deadlines with rate-monotonic scheduling



Earliest-deadline-first scheduling

Real-time CPU Scheduling

- Proportional 成比例的 Share Scheduling
  - Proportional share schedulers operate by allocating T shares among all applications
  - An application can receive N shares of time, thus ensuring that the application will have N/T of the total processor time

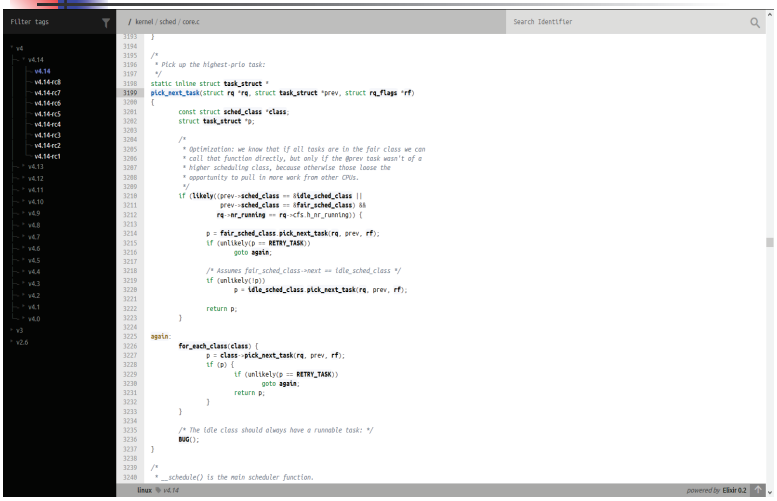
Real-time CPU Scheduling

- POSIX Real-Time Scheduling
  - SCHED_FIFO
  - SCHED_RR
  - SCHED_OTHER
  - pthread attr_getsched_policy(pthread_attr_t *attr, int *policy)
  - pthread attr_setsched_policy(pthread_attr_t *attr, int policy)

Multiple-Processor Scheduling

- multiple CPUs
  - load sharing becomes possible—but scheduling problems become correspondingly more complex
- Approaches to Multiple-Processor Scheduling
  - asymmetric multiprocessing
    - all scheduling decisions, I/O processing, and other system activities handled by a single processor—the master server.
    - The other processors execute only user code.
  - symmetric multiprocessing (SMP)
    - Each processor is self-scheduling. All processes may be in a common ready queue, or each processor may have its own private queue of ready processes.

Linux Scheduler (cont.,)



Multiple-Processor Scheduling

- Processor Affinity 处理器亲和性
  - Consider what happens to cache memory when a process has been running on a specific processor. The data most recently accessed by the process populate the cache for the processor.
  - As a result, successive memory accesses by the process are often satisfied in cache memory.
- Deference Forms of Processor Affinity
  - soft affinity
    - When an operating system has a policy of attempting to keep a process running on the same processor—but not guaranteeing that it will do so
  - hard affinity
    - sched_setaffinity() system call

Thread Scheduling

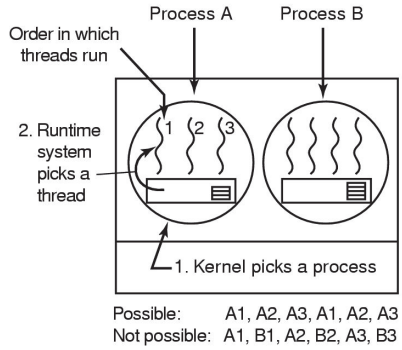
- Two levels of parallelism
- Thread scheduler
  - User-level thread
  - Kernel-level thread
  - (Hyper Thread)

Linux Scheduler

- /kernel/sched/core.c
  - static void __sched notrace __schedule(bool preempt)
  - static __always_inline struct rq *
    context_switch(struct rq *rq, struct task_struct *prev,
    struct task_struct *next, struct rq_flags *rf)
  - static inline struct task_struct *
    pick_next_task(struct rq *rq, struct task_struct *prev, struct rq_flags *rf)

Thread Scheduling

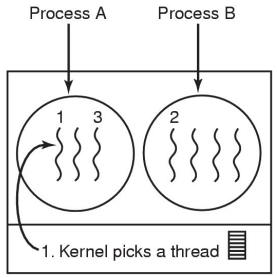
User-level thread: process-contention scope (PCS)
进程范围竞争



(a) Possible scheduling of user-level threads with a 50-msec process quantum and threads that run 5 msec per CPU burst.

Thread Scheduling

Kernel-level thread: system-contention scope (SCS)



Possible: A1, A2, A3, A1, A2, A3
Also possible: A1, B1, A2, B2, A3, B3

(b) Possible scheduling of kernel-level threads with the same characteristics as (a).

Thread Scheduling Case

```
#include <pthread.h>  #include <stdio.h>
#define NUM THREADS 5
int main(int argc, char *argv[]){
 int i;
pthread_t tid[NUM THREADS];
pthread_attr t attr;
/* get the default attributes */
pthread_attr init(&attr);
/* set the scheduling algorithm to PROCESS or SYSTEM */
pthread_attr setscope(&attr, PTHREAD_SCOPE_SYSTEM);
/* set the scheduling policy - FIFO, RT, or OTHER */
pthread attr setschedpolicy(&attr, SCHED OTHER);
/* create the threads */
for (i = 0; i < NUM THREADS; i++)
pthread_create(&tid[i],&attr,runner,NULL);
```


Thread Scheduling Case

```
/* now join on each thread */
for (i = 0; i < NUM THREADS; i++)
  pthread_join(tid[i], NULL);
}

 /* Each thread will begin control in this function */
void *runner(void *param)
{
   printf("I am a thread\n");
   pthread_exit(0);
}
```


Thread Scheduling

- Pthread Scheduling
  - PTHREAD_SCOPE_PROCESS schedules threads using PCS scheduling
  - PTHREAD_SCOPE_SYSTEM schedules threads using SCS scheduling
- pthread attr setscope(pthread attr t *attr, int scope)
- pthread attr getscope(pthread attr t *attr, int *scope)

Policy v.s. Mechanism

- Scheduling mechanism 调度机制
- Scheduling policy 调度策略

Summary

- Scheduler
- Process Behavior
- Scheduling Mode
- Scheduling Criteria
- Scheduling Algorithms
- Thread Scheduling
- ...

Q&A?

thank you
danke
謝謝
ngiyabonga
teşekkür ederim
спасибо
gracias
tapadh leat
dank je
mochchakkeram
bedankt
dziękuję
hvala
go raibh maith agat
obrigado
sukriya
kop khun krap
arigatō
takk
dakujem
sağolun
terima kasih
grazie
мерси
merci
감사합니다
ευχαριστώ