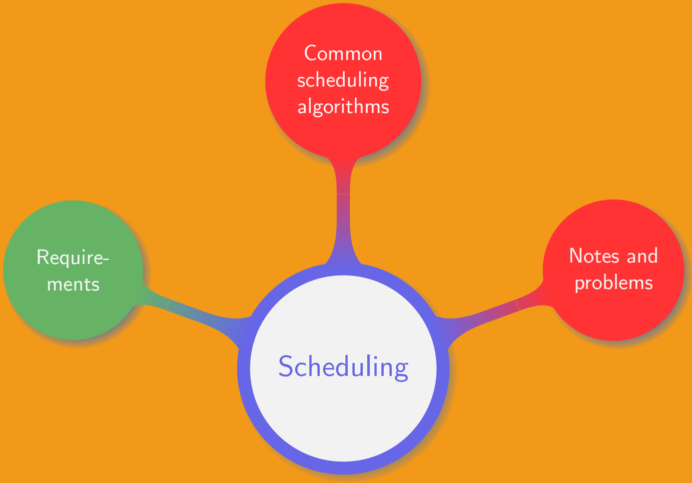
Introduction to Operating Systems

4. Scheduling

Manuel – Fall 2020

Chapter organisation



Initial problem

Scheduler's job:

- Multiple processes competing for using the CPU
- More than one process in ready state
- Which one to select next?
- Key issue in terms of "perceived performance"
- Need "clever" and efficient scheduling algorithms

Scheduler’s job

Switching process is expensive:

- Switch from user mode to kernel mode
- Save state of current process (save register, memory map…)
- Run scheduling algorithm to select a new process
- Remap the memory address for the new process
- Start new process

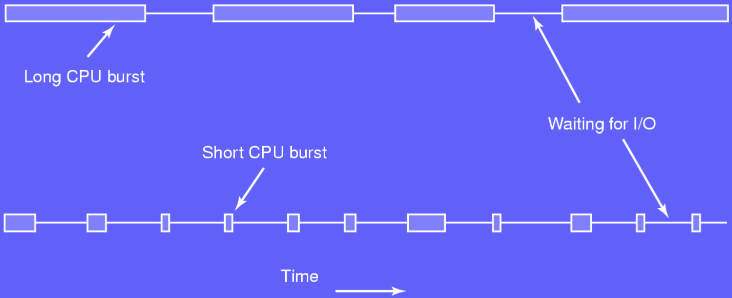
Scheduler's job

Switching process is expensive:

- Switch from user mode to kernel mode
- Save state of current process (save register, memory map...)
- Run scheduling algorithm to select a new process
- Remap the memory address for the new process
- Start new process

Too many switches per second wastes much CPU

Process behavior



Typical behavior:

- Process runs for a while
- System call emitted to read/write from/in a file
- More general: process in blocked state until external device has completed its work

Process behavior and scheduling times

Compute bound vs. input-output bound:

- Most time spent computing vs. waiting for IO
- Length of the CPU burst:
  - IO time is constant
  - Processing data is not constant
- As CPUs get faster processes are more and more IO bound

Process behavior and scheduling times

Compute bound vs. input-output bound:

- Most time spent computing vs. waiting for IO
- Length of the CPU burst:
  - IO time is constant
  - Processing data is not constant
- As CPUs get faster processes are more and more IO bound

Deciding what process to run next:

- A new process is created
- A process exits or blocks
- IO interrupt from a device that has completed its task

Process behavior and scheduling times

Compute bound vs. input-output bound:

- Most time spent computing vs. waiting for IO
- Length of the CPU burst:
  - IO time is constant
  - Processing data is not constant
- As CPUs get faster processes are more and more IO bound

Deciding what process to run next:

- A new process is created
- A process exits or blocks
- IO interrupt from a device that has completed its task

How to decide when it is best to run a process?

Preemptive vs. non-preemptive

Two main strategies for scheduling algorithms:

- Preemptive:
  - A process is run for at most *n* ms
  - If it is not completed by the end of the period then it is suspended
  - Another process is selected and run
- Non-preemptive:
  - A process runs until it blocks or voluntarily releases the CPU
  - It is resumed after an interrupt unless another process with higher priority is in the queue

Preemptive vs. non-preemptive

Two main strategies for scheduling algorithms:

- Preemptive:
  - A process is run for at most n ms
  - If it is not completed by the end of the period then it is suspended
  - Another process is selected and run
- Non-preemptive:
  - A process runs until it blocks or voluntarily releases the CPU
  - It is resumed after an interrupt unless another process with higher priority is in the queue

Which strategy is best and what is needed to use it?

Goals when scheduling

All systems:

- Fairness: fair share of the CPU for each process
- Balance: all parts of the system are busy
- Policy enforcement: follow the defined policy

Goals when scheduling

All systems:

- Fairness: fair share of the CPU for each process
- Balance: all parts of the system are busy
- Policy enforcement: follow the defined policy

Interactive systems:

- Response time: quickly process requests
- Proportionality: meet user's expectations

Goals when scheduling

Batch systems:

- Throughput: maximise the number of jobs per hour
- Turnaround time: minimise the time between submission and termination of a job
- CPU utilisation: keep the CPU as busy as possible

Goals when scheduling

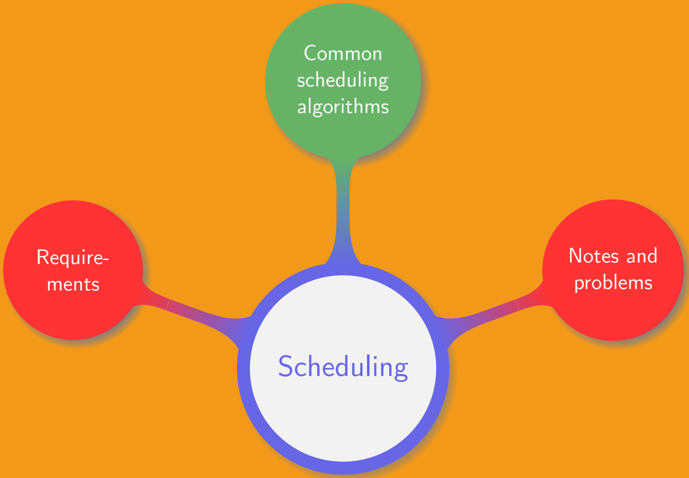
Batch systems:

- Throughput: maximise the number of jobs per hour
- Turnaround time: minimise the time between submission and termination of a job
- CPU utilisation: keep the CPU as busy as possible

Real-time systems:

- Meet deadlines: avoid any data loss
- Predictability: avoid quality degradation, e.g. for multimedia

Chapter organisation



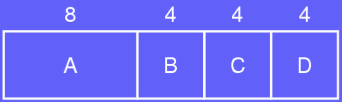
First-come, first-served

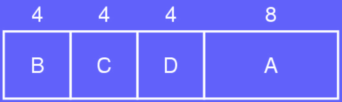
Simplest algorithm but non-preemptive:

- CPU is assigned in the order it is requested
- Processes are not interrupted, they can run a long as they want
- New jobs are put at the end of the queue
- When a process blocks the next in line is run
- Any blocked process becoming ready is pushed to the queue

When is this algorithm appropriate and when should it be avoided?

Shortest job first





Non-preemptive algorithm with all run times known in advance:

- Run time: A: 8 min, B: 4 min, C: 4 min, D: 4 min
- Turnaround time: $\frac{8+12+16+20}{4} =$ 14 min

- Run time: B: 4 min, C: 4 min, D: 4 min, A: 8 min
- Turnaround time: $\frac{4+8+12+20}{4} =$ 11 min

When is this algorithm appropriate and when should it be avoided?

Round-Robin scheduling

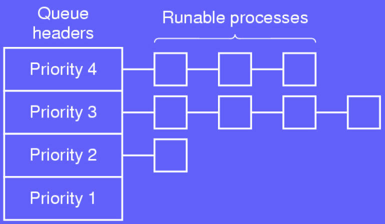
Preemptive, simple, fair, and most widely used algorithm:

- Each process is assigned a time interval called *quantum*
- A process runs until:
  - Getting blocked
  - being completed
  - Its quantum has elapsed
- A process switch occurs

When is this algorithm appropriate and when should it be avoided?

Priority scheduling

Preemptive algorithm allowing to define priorities based on who or what:



- Processes are more or less important, e.g. printing
- Creates priority classes
- Use Round-Robin within a class
- Run higher priority processes first

When is this algorithm appropriate and when should it be avoided?

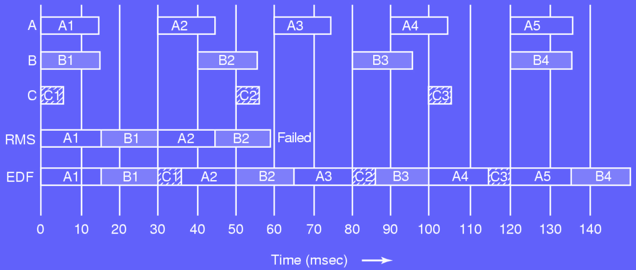
Lottery scheduling

Preemptive algorithm which can extend priority scheduling:

- Processes get lottery tickets
- When a scheduling decision is made a random ticket is chosen
- Price for the winner is to access resources
- High priority processes get more tickets

When is this algorithm appropriate and when should it be avoided?

Earliest deadline first

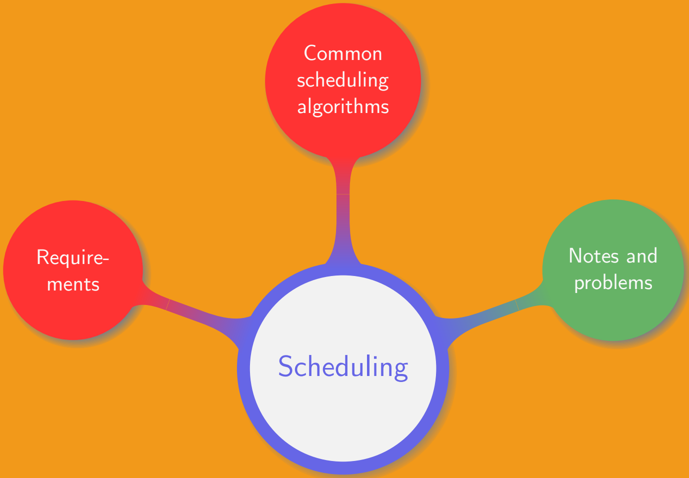


Priority based preemptive algorithm:

- Process needs to announce (i) its presence and (ii) its deadline
- Scheduler orders processes with respect to their deadline
- First process in the list (earliest deadline) is run

When is this algorithm appropriate and when should it be avoided?

Chapter organisation



Policy vs. mechanism

Limitations of the previous algorithms:

- They all assume that processes are competing
- Parent could know which of its children is most important

Policy vs. mechanism

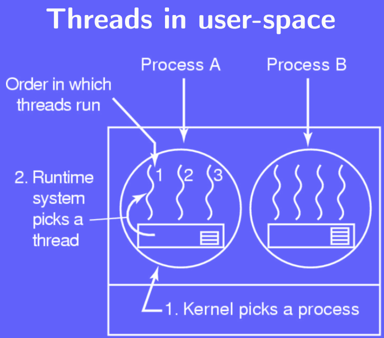
Limitations of the previous algorithms:

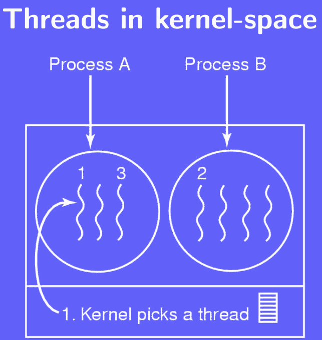
- They all assume that processes are competing
- Parent could know which of its children is most important

Separate the scheduling mechanism from the scheduling policy:

- Scheduling algorithm has parameters
- Parameters can be set by processes
- A parent can decide which of its children should have higher priority

Threads scheduling





In each case which of the following running orders are possible:

- A1, A2, A3, A1, A2, A3
- A1, B1, A2, B2, A3, B3

The dining philosophers problem

Synchronisation problem:

- A philosopher is either thinking or eating
- When he is hungry he takes:
  1. His left chop-stick
  2. His right chop-stick
- Eats
- Puts down his chop-sticks
- Thinks

The dining philosophers problem

First obvious solution:

- Wait for a chop-stick to be available
- Seize it as soon as it becomes available

The dining philosophers problem

First obvious solution:

- Wait for a chop-stick to be available
- Seize it as soon as it becomes available

What if they all take the left chop-stick at the same time?

The dining philosophers problem

First obvious solution:

- Wait for a chop-stick to be available
- Seize it as soon as it becomes available

What if they all take the left chop-stick at the same time?

Second solution:

- Take left chop-stick
- If right chopstick not available put down the left one
- Wait for some time and repeat the process

The dining philosophers problem

First obvious solution:

- Wait for a chop-stick to be available
- Seize it as soon as it becomes available

What if they all take the left chop-stick at the same time?

Second solution:

- Take left chop-stick
- If right chopstick not available put down the left one
- Wait for some time and repeat the process

What if they all start at the same time?

The dining philosophers problem

A solution using mutex:

- A philosopher thinks
- Locks mutex
- Acquires chop-sticks, eat, put them down
- Unlocks the mutex

The dining philosophers problem

A solution using mutex:

- A philosopher thinks
- Locks mutex
- Acquires chop-sticks, eat, put them down
- Unlocks the mutex

How many philosophers can eat at the same time?

The dining philosophers problem

```
#define N 5
#define LEFT (i+N-1)%N
#define RIGHT (i+1)%N
enum { THINKING, HUNGRY, EATING };
int state[N]; mutex mut = 0 ; semaphore s[N];
void philosopher(int i) {while(TRUE) {think();take_cs(i);eat();put_cs(i);}}
void take_cs(int i) {
  mutex-lock(&mut);
  state[i] = HUNGRY; test(i);
  mutex-unlock(&mut); down(&s[i]);
}
void put_cs(int i) {
  mutex-lock(&mut);
  state[i] = THINKING; test(LEFT); test(RIGHT);
  mutex-unlock(&mut);
}
void test(int i) {
  if(state[i]==HUNGRY && state[LEFT]!=EATING && state[RIGHT]!=EATING;) {
    state[i]=EATING; up(&s[i]); }
}
```

Key points

- Why is scheduling the lowest part of the OS?
- What are the two main types of algorithm?
- What are the two most common scheduling algorithms?
- Give an example of theoretical problem related to scheduling

Thank you!