Principper for Samtidighed og Styresystemer

Scheduling

René Rydhof Hansen

25 FEB 2019

Next time (04 MAR 2019): XV6 Hands-on Exercises

- No lecture
- TAs: 14:30–16:15 (approx)
- XV6 exercises (in group rooms)

Learning Goals (last time)

After last lecture you

- ... can define and explain the concept of a process
- ... can explain what a process image is
- ... can explain what a process control block, what it is used for and why it is needed
- ... can explain, in general terms, how process creation, switching and termination works
- ... can define and discuss process states
- ... can define and explain the concept of a thread
- ... can discuss when, where, and how (multi-)threads are useful
- ... can explain what a thread control block, what it is used for and why it is needed
- ... can discuss implementation strategies for thread support and explain the associated trade-offs

Learning Goals

After today's lecture you

- ... can explain the notion of limited direct execution and how it relates to scheduling
- ... will know the simplified process model
- ... will know and can explain important metrics for measuring a scheduling policy:
  - Fairness
  - Turnaround time
  - Response time
- ... can explain important scheduling policies and their pros and cons
  - FIFO
  - SJF
  - STCF
  - Round robin
  - MLFQ
  - Lottery scheduling

Limited Direct Execution

How to optimise CPU usage?

Multitasking

- Processes share the CPU

Problem(s)?

- Efficient execution of program
- Retaining control of running program
  - How to avoid dangerous/unwanted behaviour?
  - How to stop currently running program?

The Answer: Limited Direct Execution

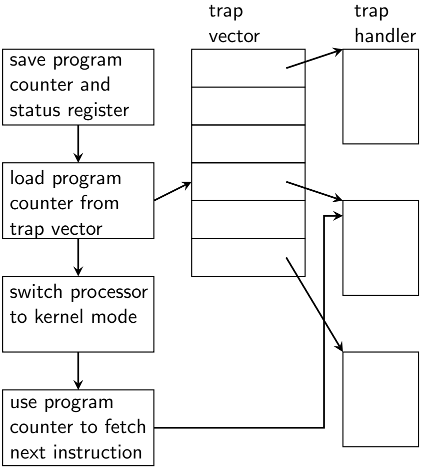
- User- vs- kernel-space (hw support)
- Timer interrupts (hw support)

Recall from last time: Running a Program (v4)

1. Load code from secondary memory
2. Allocate memory for the process
3. Start executing
4. Continue until
   1. Program executes blocking operation
   2. Time slice expires
5. Switch to another program
   1. Save the relevant data for current program
   2. Pick highest-priority program, ready to run
   3. Re-instate new program's data
   4. Execute

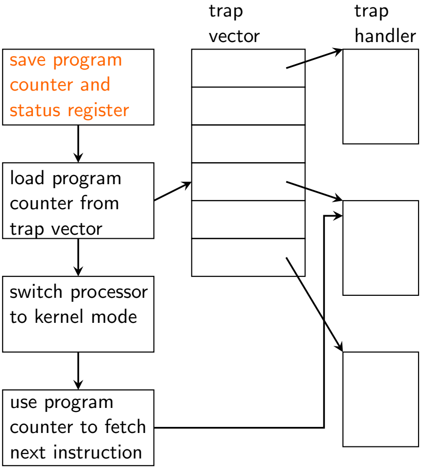
Traps: Handling/implementation

- Trap handler called
  - Program counter and status register saved on stack
  - Program counter set to address of exception handler
  - Kernel switches to kernel-mode
- Further information is saved
  - Current processor mode
  - Information about cause of exception
- Trap vector
  - Table with addresses of traps handlers
  - Initialised by operating system



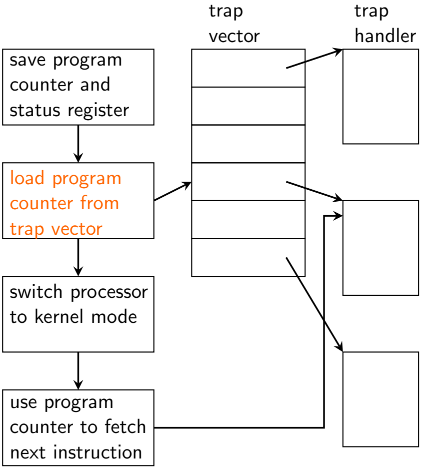
Traps: Handling/implementation

- Trap handler called
  - Program counter and status register saved on stack
  - Program counter set to address of exception handler
  - Kernel switches to kernel-mode
- Further information is saved
  - Current processor mode
  - Information about cause of exception
- Trap vector
  - Table with addresses of traps handlers
  - Initialised by operating system



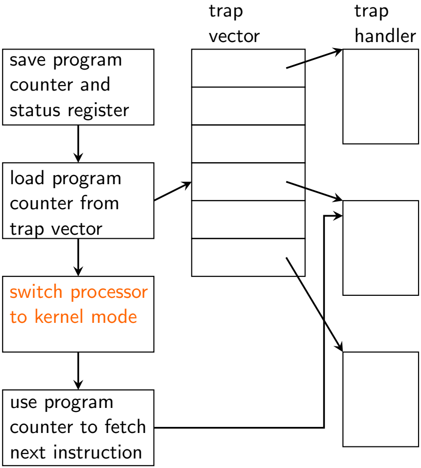
Traps: Handling/implementation

- Trap handler called
  - Program counter and status register saved on stack
  - Program counter set to address of exception handler
  - Kernel switches to kernel-mode
- Further information is saved
  - Current processor mode
  - Information about cause of exception
- Trap vector
  - Table with addresses of traps handlers
  - Initialised by operating system



Traps: Handling/implementation

- Trap handler called
  - Program counter and status register saved on stack
  - Program counter set to address of exception handler
  - Kernel switches to kernel-mode
- Further information is saved
  - Current processor mode
  - Information about cause of exception
- Trap vector
  - Table with addresses of traps handlers
  - Initialised by operating system



Simple Scheduling

Scheduling: Introduction

Simplified process model

1. Each job runs for the same amount of time.
2. All jobs arrive at the same time.
3. Once started, each job runs to completion.
4. All jobs only use the CPU (i.e., they perform no I/O)
5. The run-time of each job is known.

Metric: Turnaround Time

A metric for "how good" a scheduler is

$$T_{turnaround} = T_{completion} - T_{arrival}$$

Terminology

For scheduling: job == process || job == thread

Scheduling: FIFO

FIFO

- Jobs are scheduled in the order they arrive

Pros and Cons of FIFO Scheduling

- Easy to implement
- Little overhead
- Blissfully ignorant
- Fair
- "Convoy effect"

Example (FIFO turnaround time)

Average turnaround for $T^i_{exec} = 10$ $(i = 1, 2, 3)$?

Relaxing Assumption #1

Assumptions: Simplified process model

1. Each job runs for the same amount of time.
2. All jobs arrive at the same time.
3. Once started, each job runs to completion.
4. All jobs only use the CPU (i.e., they perform no I/O)
5. The run-time of each job is known.

Relaxing Assumption #1

Assumptions: Simplified process model

1. Each job runs for the same amount of time.
2. All jobs arrive at the same time.
3. Once started, each job runs to completion.
4. All jobs only use the CPU (i.e., they perform no I/O)
5. The run-time of each job is known.

Example (FIFO turnaround time (re-visited))

Average turnaround for $T^1_{exec} = 10$, $T^2_{exec} = 10$, $T^3_{exec} = 100$? This is called the "convoy effect"

Scheduling: Shortest Job First (SJF)

Shortest Job First (SJF)

- Jobs are sorted according to run-time length
- Shortest (wrt. run-time) job is executed first

Pros and Cons of SJF Scheduling

- Provably optimal
- Requires sorting of processes
- Fair?

Scheduling: Shortest Job First (SJF)

Shortest Job First (SJF)

- Jobs are sorted according to run-time length
- Shortest (wrt. run-time) job is executed first

Pros and Cons of SJF Scheduling

- Provably optimal
- Requires sorting of processes
- Fair?

Example (SJF turnaround time)

Average turnaround for $T^1_{exec} = 10,\ T^2_{exec} = 10,\ T^3_{exec} = 100$?

Relaxing Assumption #2

Assumptions: Simplified process model

1. Each job runs for the same amount of time
2. All jobs arrive at the same time
3. Once started, each job runs to completion
4. All jobs only use the CPU (i.e., they perform no I/O)
5. The run-time of each job is known

Relaxing Assumption #2

Assumptions: Simplified process model

1. Each job runs for the same amount of time
2. All jobs arrive at the same time
3. Once started, each job runs to completion
4. All jobs only use the CPU (i.e., they perform no I/O)
5. The run-time of each job is known

Problem with SJF

- What if short(er) jobs arrive later?

Example (SJF turnaround time (re-visited))

Average turnaround for $T^1_{exec} = 10$, $T^2_{exec} = 10$, $T^3_{exec} = 100$, with $T^1_{arrival} \geq T^2_{arrival} > T^3_{arrival}$?

Relaxing Assumption #3

Assumptions: Simplified process model

1. Each job runs for the same amount of time
2. All jobs arrive at the same time
3. Once started, each job runs to completion
4. All jobs only use the CPU (i.e., they perform no I/O)
5. The run-time of each job is known

Relaxing Assumption #3

Assumptions: Simplified process model

1. Each job runs for the same amount of time
2. All jobs arrive at the same time
3. Once started, each job runs to completion
4. All jobs only use the CPU (i.e., they perform no I/O)
5. The run-time of each job is known

Preemption

- OS can preempt running process and (maybe) switch to new process
- Process-switch is expensive (overhead)

Scheduling: Shortest Time-to-Completion First (STCF)

Shortest Time-to-Completion First (STCF)

Whenever a (new) process enters the scheduler:

- Currently running process is pre-empted
- Jobs are sorted according to run-time left
- Job with shortest remaining run-time is allowed to run

Pros and Cons of STCF

- Fair?
- Provably optimal

Scheduling: Shortest Time-to-Completion First (STCF)

Shortest Time-to-Completion First (STCF)

Whenever a (new) process enters the scheduler:

- Currently running process is pre-empted
- Jobs are sorted according to run-time left
- Job with shortest remaining run-time is allowed to run

Pros and Cons of STCF

- Fair?
- Provably optimal
  - ... *if* execution time (job length) is known
  - ... *if* only turnaround is used
  - ... *if* no blocking operations are performed

Response time

Why turnaround time?

- Good for batch systems (optimises throughput)
- Good for interactive systems (latency)?

Metric: Response time

$$T_{response} = T_{firstrun} - T_{arrival}$$

- Note: in RTS response time is defined more like turnaround-time

Response time for FIFO, SJF, STFC

???

Scheduling: Round Robin

Round Robin

- Processes are entered into a circular queue
- Each process runs for a short while (time slice)
- OS switches to next process

Pros and Cons of RR

- Fair?
- Poor turnaround time
- Good response time
- Tuning size of time slice (Warning: here be dragons!)

Scheduling: Round Robin

Round Robin

- Processes are entered into a circular queue
- Each process runs for a short while (time slice)
- OS switches to next process

Pros and Cons of RR

- Fair?
- Poor turnaround time
- Good response time
- Tuning size of time slice (Warning: here be dragons!)

Example (SJF vs RR response time)

Average response time for $T^i_{exec} = 5$, (for $i = 1, 2, 3$) under SJF and RR (time slice $=$ 1 time unit)

Relaxing Assumptions #4 and #5

Assumptions: Simplified process model

1. Each job runs for the same amount of time
2. All jobs arrive at the same time
3. Once started, each job runs to completion
4. All jobs only use the CPU (i.e., they perform no I/O)
5. The run-time of each job is known

- Unrealistic(?): run-time known beforehand
  - (Hard-)RTS: necessary to know beforehand
- Unrealistic(?): No I/O

Overlap

- Switch to ready process while waiting for I/O

Relaxing Assumptions #4 and #5

Assumptions: Simplified process model

1. Each job runs for the same amount of time
2. All jobs arrive at the same time
3. Once started, each job runs to completion
4. All jobs only use the CPU (i.e., they perform no I/O)
5. The run-time of each job is known

Overlap

- Switch to ready process while waiting for I/O

Not so simple scheduling

Scheduling with priorities

Scheduling goals

- Goal: optimise turnaround time
- Goal: minimise response time

The Multi-Level Feedback Queue

- Assign priorities to processes
- Each priority level has its own process queue

The Rules (MLFQ): How to schedule

Rule 1 If $Pri(A) > Pri(B)$ run A

Rule 2 If $Pri(A) = Pri(B)$ use RR for A and B

Multi-Level Feedback Queue

Changing priorities

- Claim: response time not/less important for CPU bound jobs
- Claim: jobs that often release CPU (before time): “interactive”

The Rules (MLFQ): Changing priorities

Multi-Level Feedback Queue

Changing priorities

- Claim: response time not/less important for CPU bound jobs
- Claim: jobs that often release CPU (before time): "interactive"

The Rules (MLFQ): Changing priorities

- Rule 3: New processes are given highest priority
- Rule 4a: Priority of process using all its time slice is reduced
- Rule 4b: A process releasing the CPU before time, stays at the same priority

Problems with the MLFQ

Starvation

- What happens when many interactive (short) processes must be scheduled?

Problems with the MLFQ

Starvation

- What happens when many interactive (short) processes must be scheduled?
- Solution: the priority boost

The Rules (MLFQ)

- **Rule 1** If $Pri(A) > Pri(B)$ run A
- **Rule 2** If $Pri(A) = Pri(B)$ use RR for A and B
- **Rule 3** New processes are given highest priority
- **Rule 4a** Priority of process using all its time is reduced
- **Rule 4b** A process releasing the CPU before time, stays at the same priority
- **Rule 5** "Boost" all processes at a fixed timed-interval

Problems with the MLFQ

Gaming the System

- What happens if a process uses almost all its time and then makes a system call? Repeatedly?

The Rules (MLFQ)

Rule 1 If $Pri(A) > Pri(B)$ run A

Rule 2 If $Pri(A) = Pri(B)$ use RR for A and B

Rule 3 New processes are given highest priority

Rule 5 “Boost” all processes at a fixed timed-interval

Problems with the MLFQ

Gaming the System

- What happens if a process uses almost all its time and then makes a system call? Repeatedly?

The Rules (MLFQ)

Rule 1 If $Pri(A) > Pri(B)$ run A

Rule 2 If $Pri(A) = Pri(B)$ use RR for A and B

Rule 3 New processes are given highest priority

Rule 4 Always reduce priority of a process when alloted time has been used

Rule 5 “Boost” all processes at a fixed timed-interval

Proportional Share Scheduling

Proportional Share Scheduling

Fair share of CPU

- Optimise for CPU time (not response/turnaround time)

The Lottery Scheduler

- Assign (a number of) tickets to each process
- Draw a (random) ticket
- Process with that ticket gets to run

"Interesting" Features

- Probabilistic (no hard guarantees, only statistical)
- Trivial to implement
- Processes can trade tickets (why?)

Non-deterministic fair-share scheduler

Stride Scheduling

- Each process is assigned a stride inversely proportional to number of tickets
- Each aprocess is assigned a pass number, incremented by the stride for every run
- The process with the lowest pass values runs

Learning Goals

After today's lecture you

- ... can explain the notion of limited direct execution and how it relates to scheduling
- ... will know the simplified process model
- ... will know and can explain important metrics for measuring a scheduling policy:
    - Fairness
    - Turnaround time
    - Response time
- ... can explain important scheduling policies and their pros and cons
    - FIFO
    - SJF
    - STCF
    - Round robin
    - MLFQ
    - Lottery scheduling