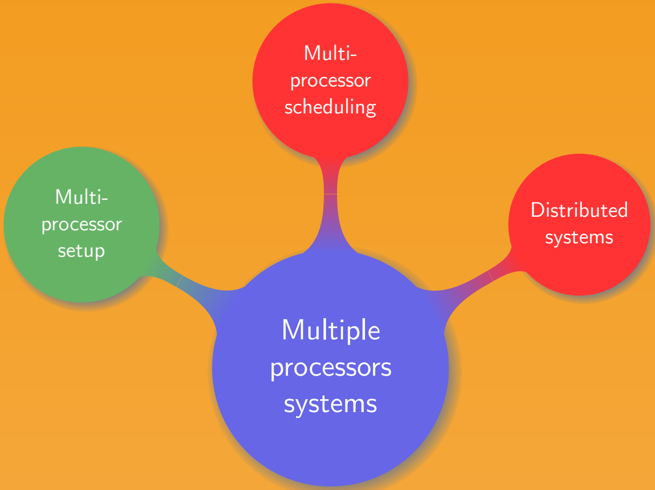
Introduction to Operating Systems

10. Multiple processors systems

Manuel – Fall 2019

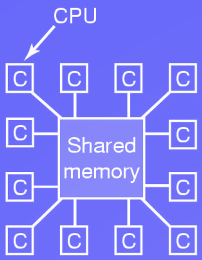
Chapter organisation



Shared memory model

Multiprocessors:

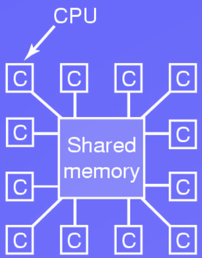
- CPUs communicate through the shared memory
- Every CPU has equal access to entire physical memory
- Access time: 2-10 ns



Shared memory model

Multiprocessors:

- CPUs communicate through the shared memory
- Every CPU has equal access to entire physical memory
- Access time: 2-10 ns



Three main approaches:

- Each CPU has its own OS: no sharing, all independent
- Master-slave multiprocessors: one CPU handles all the requests
- Symmetric Multi-Processor: solution used in practice

SMP approach

One copy of the OS that can be run by any of the CPUs

Problem: what if two CPUs run the same process or claim the same free memory page at the same time?

SMP approach

One copy of the OS that can be run by any of the CPUs

Problem: what if two CPUs run the same process or claim the same free memory page at the same time?

Solution:

- Many parts of the OS are independent
- Split the OS into multiple critical regions
- Add a mutex when entering those regions
- Add mutex to all shared tables

SMP approach

One copy of the OS that can be run by any of the CPUs

Problem: what if two CPUs run the same process or claim the same free memory page at the same time?

Solution:

- Many parts of the OS are independent
- Split the OS into multiple critical regions
- Add a mutex when entering those regions
- Add mutex to all shared tables

Challenges:

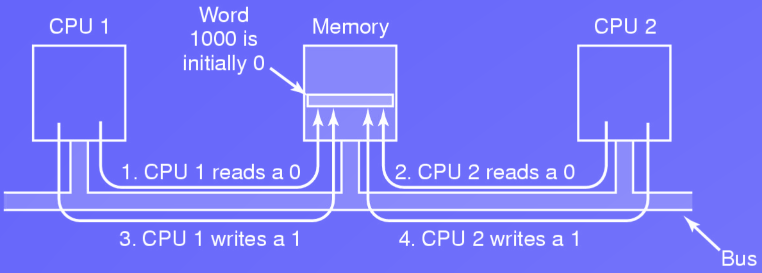
- How to divide up the OS
- Easy to run into deadlock with the shared tables
- Hard to keep consistency between programmers

Synchronisation – Problem

Synchronisation strategy with a single CPU:

Synchronisation – Problem

Synchronisation strategy with a single CPU: `TSL` instruction



Synchronisation – Solution

The `TSL` instruction should:

1. Lock the bus by asserting a special line on the bus
2. Test and Set the Lock
3. Unlock the bus

Synchronisation – Solution

The `TSL` instruction should:

1. Lock the bus by asserting a special line on the bus
2. Test and Set the Lock
3. Unlock the bus

New multiprocessor issue: slows down the whole system

Solution: use a local cache such as not to block the bus to often

Cache with multiprocessors

Multiprocessors cache implementation:

- Requesting CPU reads the lock and gets a copy in its cache
- Polling is done using the value in cache
- When the lock is remove the cache protocol invalidates all the remote copies
- Updated value is to be fetched

Cache with multiprocessors

Multiprocessors cache implementation:

- Requesting CPU reads the lock and gets a copy in its cache
- Polling is done using the value in cache
- When the lock is remove the cache protocol invalidates all the remote copies
- Updated value is to be fetched

Problem:

- Mutex is 1 bit, but a whole block is copied
- `TSL` require write access
- Cached block that is modified is invalidated
- Cache need to be recopied

Read and TSL

New approach:

- Check if the lock is free using a read
- If lock appears to be free apply `TSL` instruction

Read and TSL

New approach:

- Check if the lock is free using a read
- If lock appears to be free apply `TSL` instruction

Problem: what if two CPUs see the lock being free and apply the `TSL` instruction? Does it lead to a race condition?

Read and TSL

New approach:

- Check if the lock is free using a read
- If lock appears to be free apply `TSL` instruction

Problem: what if two CPUs see the lock being free and apply the `TSL` instruction? Does it lead to a race condition?

- The value returned by the read is only a hint
- Only one CPU gets the lock
- The `TSL` instruction prevent any race condition

More solutions

Ethernet binary exponential back-off algorithm:

- Do not poll at regular intervals
- Add a loop where waiting time is doubled at each iteration
- Setup a maximum waiting time

More solutions

Ethernet binary exponential back-off algorithm:

- Do not poll at regular intervals
- Add a loop where waiting time is doubled at each iteration
- Setup a maximum waiting time

Set a mutex for each CPU requesting the lock:

- When a CPU requests a lock it attaches itself at the end of the list of CPUs requesting the lock
- When the initial lock is released it frees the lock of the first CPU on the list
- The first CPU enters the critical region, does its work and releases the lock
- Next CPU on the list can start its work

Spinning or switching?

New perspective:

- On a uniprocessor: the time spent on waiting is wasted
- On a multiprocessor: one CPU is waiting while another works

Dilemma: switching is expensive but looping is a waste

Best choice:

- Only know which solution was best after
- Impossible to have an always accurate optimal decision

Spinning or switching?

New perspective:

- On a uniprocessor: the time spent on waiting is wasted
- On a multiprocessor: one CPU is waiting while another works

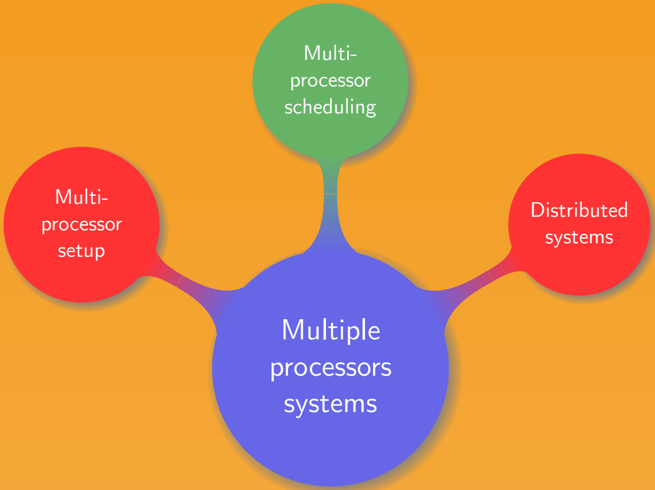
Dilemma: switching is expensive but looping is a waste

Best choice:

- Only know which solution was best after
- Impossible to have an always accurate optimal decision

Solution: mix of waiting and switching with a (variable) threshold

Chapter organisation



Setup

Single threaded process: only need to schedule the process

Uniprocessor: which thread to run?

Multiprocessor:

- Which thread to run?
- Which CPU to run it on?

Setup

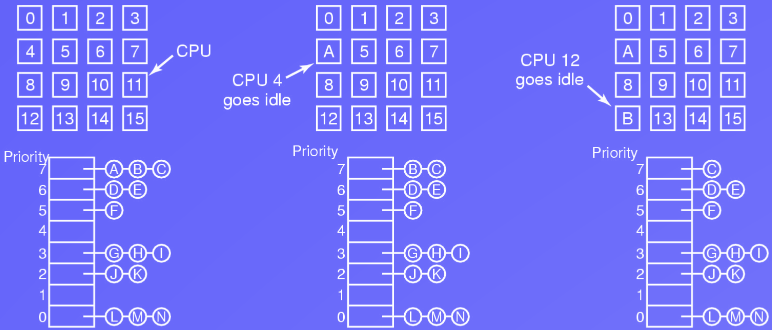
Single threaded process: only need to schedule the process

Uniprocessor: which thread to run?

Multiprocessor:

- Which thread to run?
- Which CPU to run it on?
- Threads of a process run on the same CPU $\rightarrow$ no need to reload the whole process
- Threads of a process run in parallel $\rightarrow$ threads can cooperate more efficiently

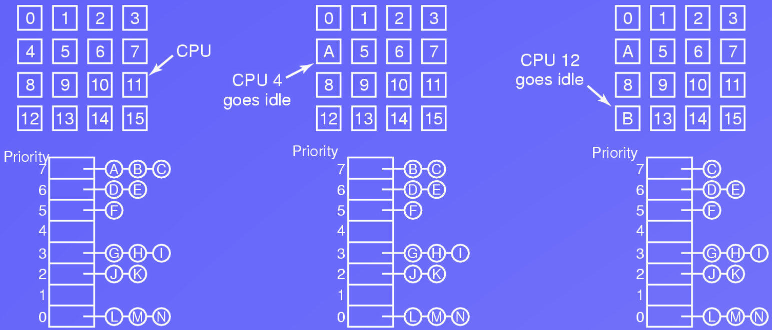
Time-sharing



- Single data structure for ready processes
- Simple and efficient implementation

Limitation: a thread holds a spin lock but reaches the end of its quantum

Time-sharing



- Single data structure for ready processes
- Simple and efficient implementation

Limitation: a thread holds a spin lock but reaches the end of its quantum → other CPUs spin waiting for lock to be released

Improvements on time-sharing

Smart scheduling:

- A thread holding a spin lock sets a flag
- Scheduler lets such thread run after the end of the quantum
- Clear the flag when lock is released

Affinity scheduling:

- Thread is assigned a CPU when it is created
- Try as much as possible to run a thread on the same CPU
- Cache affinity is maximized

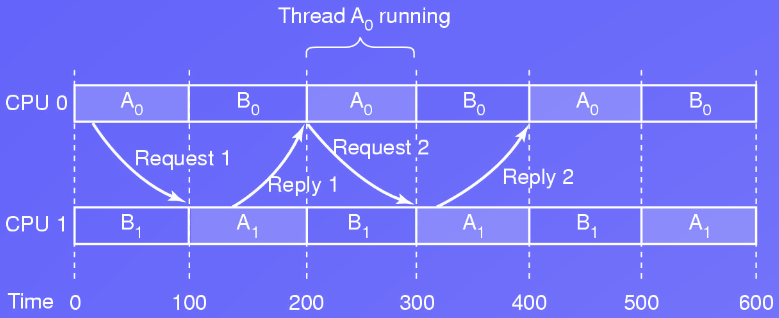
Space-sharing

When a process is created the scheduler checks if the number of free CPUs is larger then the number of threads:

- Yes: run one thread per CPU until completion
- No: wait for more free CPUs
- Keep the CPU even during I/O

Optimization: try to apply shortest job first $\rightarrow$ in practice FIFO better

The communication problem



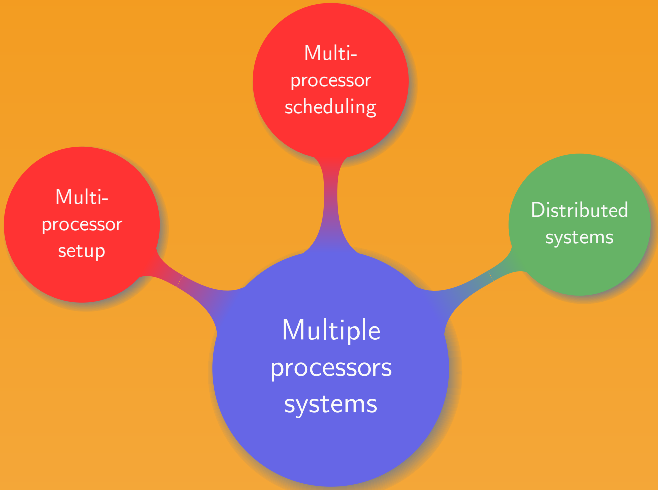
Gang scheduling

Simple idea: schedule processes by group:

- Group related threads into a gang
- All gang members run simultaneously on different CPUs
- All members start and end at the same time
- No intermediary scheduling decision is taken

Limitations: what if one gang member issue an I/O request or finishes earlier than others?

Chapter organisation



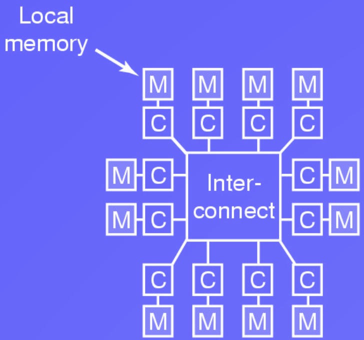
Main characteristics

Characteristics of distributed systems:

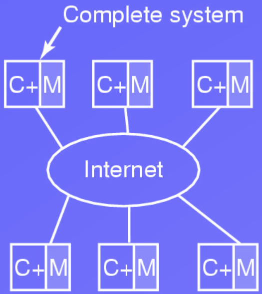
- Composed of autonomous entities, each with its own memory
- Communication is done over a network using message passing
- The system must tolerate node failures
- All the nodes perform the same task

Cluster: set of connected computers working together

Structure



Multicomputer



Distributed system

Setup and challenges

General idea of how a cluster works:

- Computing nodes connected over a LAN
- A clustering middleware sits on the top of the node
- Users view a large computer

Example.
A single master node handles the management of the scheduling and slave nodes

Main challenges:

- Scheduling: where should a job be scheduled?
- Load balancing: should a job be rescheduled on a another node?

Example

Apache Hadoop:

- Opensource framework for distributed file system
- Written in Java
- Very large files stored across multiple nodes
- Used and enhanced by Yahoo!, Facebook, Amazon, Microsoft, Google, IBM...

More on distributed computing

Advances in network technologies lead to the development of:

- Volunteer computing: volunteer offer part of their computational power to some project
- Grids: collection of computer resources from multiple locations to reach a common goal
- Jungle computing: network not necessarily composed of “regular computers”

Key points

- What is the main difference between a distributed system and a multiprocessor system?
- Explain the SMP approach
- How is `TSL` working for multiprocessors?
- Cite three scheduling strategies targeting multiprocessors
- What is a middleware?

Thank you!