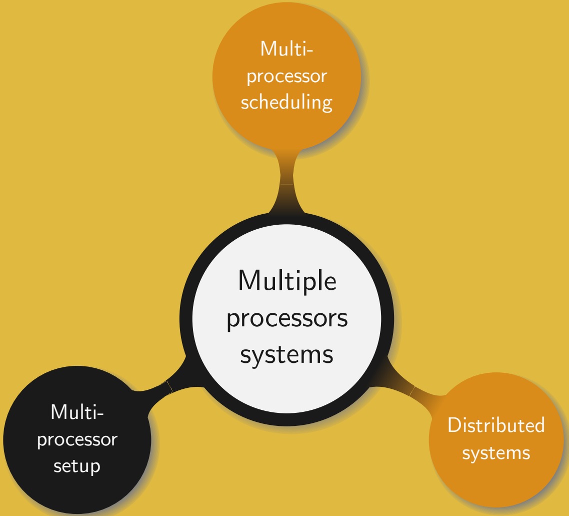
Introduction to Operating Systems

10. Multiple processors systems

Manuel – Fall 2021

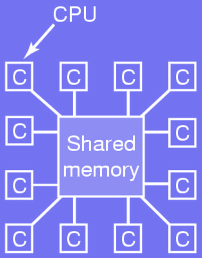
Chapter organisation



Shared memory model

Multiprocessors:

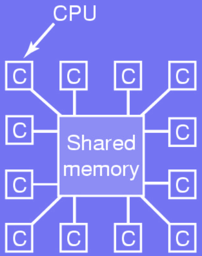
- CPUs communicate through the shared memory
- Every CPU has equal access to entire physical memory
- Access time: 2-10 ns



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: one OS shared by all CPU

SMP approach

Problems likely to occur:

- More than one CPUs run the same process
- A same free memory page is claimed at the same time

SMP approach

Problems likely to occur:

- More than one CPUs run the same process
- A same free memory page is claimed at the same time

Basic idea of the 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

Problems likely to occur:

- More than one CPUs run the same process
- A same free memory page is claimed at the same time

Basic idea of the 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

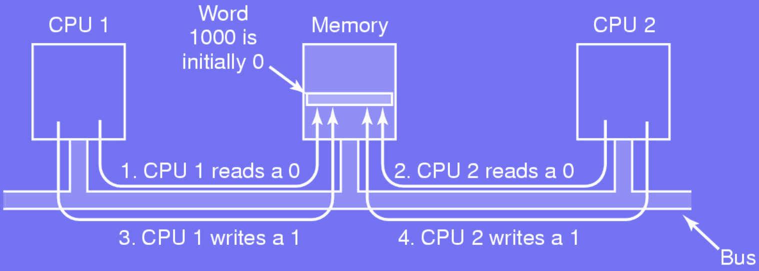
The solution works but add complexity to the OS:

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

Synchronisation

Synchronisation strategies:

- Single CPU:
  - Kernel space: disable interrupts
  - User space: take advantage of atomic operations, e.g. mutex
- Multiple CPUs:
  - Disabling interrupts on the current CPU is not enough
  - The memory bus needs to be disabled



Synchronisation

A first idea:

1. Lock the memory bus by asserting a special line on the bus
2. The bus is only available to the processor which locked it
3. Perform the necessary memory accesses
4. Unlock the bus

Synchronisation

A first idea:

1. Lock the memory bus by asserting a special line on the bus
2. The bus is only available to the processor which locked it
3. Perform the necessary memory accesses
4. Unlock the bus

New multiprocessor issue: slows down the whole system

Solution: do not lock the whole bus, but just the cache

Cache with multiprocessors

Multiprocessors cache implementation:

- The requesting CPU reads the lock and copies it in its cache
- As long as the lock is unchanged the cached value can be used
- When the lock is released all the remote copies are invalidated
- All other CPUs must fetch the updated value

Cache with multiprocessors

Multiprocessors cache implementation:

- The requesting CPU reads the lock and copies it in its cache
- As long as the lock is unchanged the cached value can be used
- When the lock is released all the remote copies are invalidated
- All other CPUs must fetch the updated value

Problem:

- The `TSL` instruction is used to acquire the lock
- The `TSL` instruction requires write access
- Any modified cached block must be invalidated
- A whole cache block needs to be recopied each time
- Much traffic is generated just to check the lock

Read and TSL

Improved approach:

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

If two CPUs see the lock being free and apply the TSL *instruction does it lead to a race condition?*

Read and TSL

Improved approach:

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

If two CPUs see the lock being free and apply the TSL *instruction does it lead to a race condition?*

A safe solution:

- The value returned by the read is only a hint
- Only one CPU can get the lock
- The TSL instruction prevents 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
- The following 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
- Switching is expensive but looping is a waste

There is no optimal solution:

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

Spinning or switching?

New perspective:

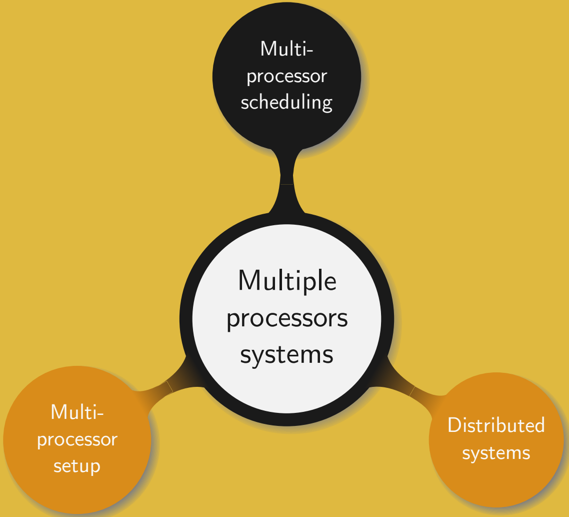
- On a uniprocessor: the time spent on waiting is wasted
- On a multiprocessor: one CPU is waiting while another works
- Switching is expensive but looping is a waste

There is no optimal solution:

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

Mix of waiting and switching with a (variable) threshold

Chapter organisation



Multi-processor scheduling

Multiple processors systems

Multi-processor setup

Distributed systems

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

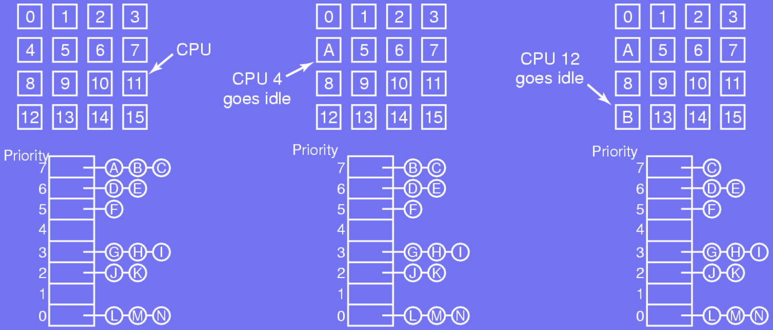
Single threaded process: only need to schedule the process

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 → no need to reload the whole process
- Threads of a process run in parallel → threads can cooperate more efficiently

Time-sharing



Advantages:

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

What if a thread holds a spin lock but reaches the end of its quantum?

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

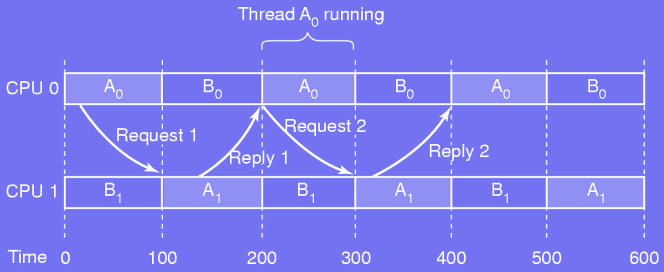
Space-sharing

When a process is created the scheduler checks if there are:

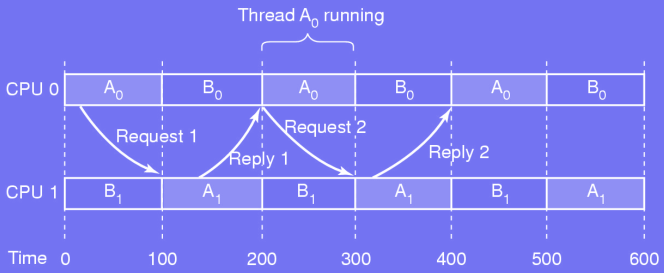
- More free CPU than threads: run a thread per CPU until completion
- More threads than free CPU: wait for more free CPU

What is the major drawback of this approach?

Gang scheduling



Gang scheduling

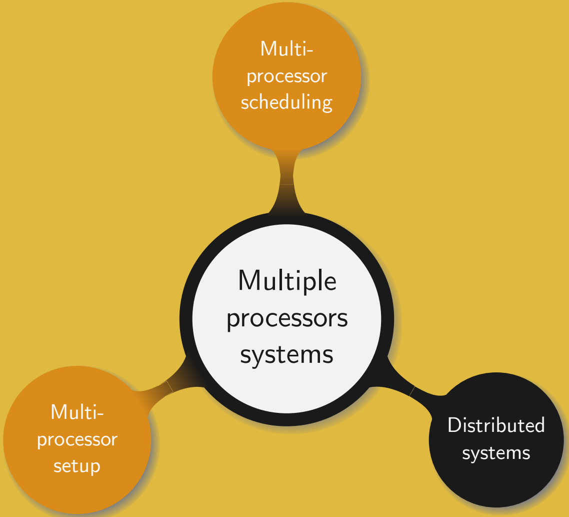


Gang scheduling schedules processes by group:

- 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

What if a member issues an IO request or finishes before others?

Chapter organisation



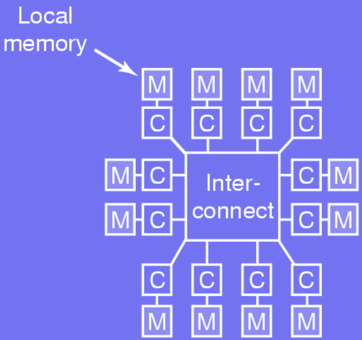
Main characteristics

Characteristics of distributed systems:

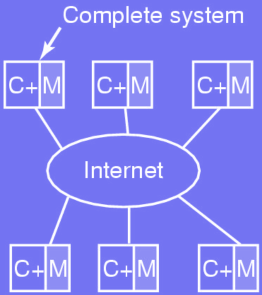
- Composed of autonomous entities, each with its own memory
- Communication is done over a network using message passing
- The system must tolerate node failures
- All the nodes perform the same task

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 handling the scheduling on slave nodes

Main challenges:

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

Interprocess communication

Over a distributed system semaphores and mutexes cannot be use

Message passing strategy:

- `send(destination,&message)`
- `receive(source,&message)`: blocks or exit if nothing is received

Potential issues:

- Messages can get lost, e.g. sending or acknowledging reception
- Confusion on the process names
- Security, e.g. authentication, traffic encryption
- Performance

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!