操作系统原理

第四章：进程管理

洪明坚

重庆大学软件学院

February 19, 2016

目录

Outline

Process concept (1/3)

Process concept (1/3)

- Before the advent of process

Process concept (1/3)

- Before the advent of process
  - Batch system

Process concept (1/3)

- Before the advent of process
  - Batch system - **job**;

Process concept (1/3)

- Before the advent of process
  - Batch system - **job**;
  - Multiprogramming or time-sharing

Process concept (1/3)

- Before the advent of process
  - Batch system - **job**;
  - Multiprogramming or time-sharing - **program or task**.

Process concept (1/3)

- Before the advent of process
  - Batch system - **job**;
  - Multiprogramming or time-sharing - **program or task**.
- **Process**

Process concept (1/3)

- Before the advent of process
  - Batch system - **job**;
  - Multiprogramming or time-sharing - **program or task**.
- **Process**
  - **An abstraction of a running job/program/task.**

Process concept (2/3)

Process concept (2/3)

- What's a process?

Process concept (2/3)

- What's a process?
  - A program in execution.

Process concept (2/3)

- What's a process?
  - A program in execution.
- A process is MUCH more than a program. It consists of

Process concept (2/3)

- What's a process?
  - A program in execution.
- A process is MUCH more than a program. It consists of
  - **Text section** (executable machine codes) from the program file;

Process concept (2/3)

- What's a process?
  - A program in execution.
- A process is MUCH more than a program. It consists of
  - **Text section** (executable machine codes) from the program file;
  - **Data section** (the global variables) from the program file;

Process concept (2/3)

- What's a process?
  - A program in execution.
- A process is MUCH more than a program. It consists of
  - **Text section** (executable machine codes) from the program file;
  - **Data section** (the global variables) from the program file;
  - *Contents* of the processor's **registers**;

Process concept (2/3)

- What's a process?
  - A program in execution.
- A process is MUCH more than a program. It consists of
  - **Text section** (executable machine codes) from the program file;
  - **Data section** (the global variables) from the program file;
  - *Contents* of the processor's **registers**;
  - **Stack** which contains temporary data such as function parameters, return addresses and local variables;

Process concept (2/3)

- What's a process?
  - A program in execution.
- A process is MUCH more than a program. It consists of
  - **Text section** (executable machine codes) from the program file;
  - **Data section** (the global variables) from the program file;
  - *Contents* of the processor's **registers**;
  - **Stack** which contains temporary data such as function parameters, return addresses and local variables;
  - **Heap** which is the memory used to be allocated dynamically if necessary;

Process concept (2/3)

- What's a process?
    - A program in execution.
- A process is MUCH more than a program. It consists of
    - **Text section** (executable machine codes) from the program file;
    - **Data section** (the global variables) from the program file;
    - *Contents* of the processor's **registers**;
    - **Stack** which contains temporary data such as function parameters, return addresses and local variables;
    - **Heap** which is the memory used to be allocated dynamically if necessary;
    - A lot of other resources such as open files, etc.

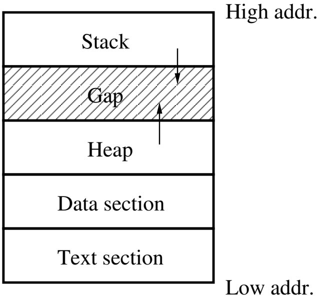
Process concept (3/3)

Process concept (3/3)

- **Process image** in memory

Process concept (3/3)

- **Process image** in memory



Program v.s. process

Program v.s. process

- The program is a **passive** entity which resides in some storage;

Program v.s. process

- The program is a **passive** entity which resides in some storage;
  - While the process is an **active** entity which contains a lot of resources other than the program.

Program v.s. process

- The program is a **passive** entity which resides in some storage;
  - While the process is an **active** entity which contains a lot of resources other than the program.
- Many processes may be running the same program. But,

Program v.s. process

- The program is a **passive** entity which resides in some storage;
  - While the process is an **active** entity which contains a lot of resources other than the program.
- Many processes may be running the same program. But,
  - They are considered as separate execution sequences although they share the same text section;

Program v.s. process

- The program is a **passive** entity which resides in some storage;
  - While the process is an **active** entity which contains a lot of resources other than the program.
- Many processes may be running the same program. But,
  - They are considered as separate execution sequences although they share the same text section;
  - Other resources usually vary.

Process state

Process state

- As a process executes, it changes **state**.

Process state

- As a process executes, it changes **state**.
  - The state of the process is defined by its current activity.

Process state

- As a process executes, it changes **state**.
    - The state of the process is defined by its current activity.
- Each process may be in one of the following states:

Process state

- As a process executes, it changes **state**.
  - The state of the process is defined by its current activity.
- Each process may be in one of the following states:
  - **New**

Process state

- As a process executes, it changes **state**.
  - The state of the process is defined by its current activity.
- Each process may be in one of the following states:
  - **New** - The process is being created;

Process state

- As a process executes, it changes **state**.
  - The state of the process is defined by its current activity.
- Each process may be in one of the following states:
  - **New** - The process is being created;
  - **Running**

Process state

- As a process executes, it changes **state**.
  - The state of the process is defined by its current activity.
- Each process may be in one of the following states:
  - **New** - The process is being created;
  - **Running** - Instructions are being executed;

Process state

- As a process executes, it changes **state**.
  - The state of the process is defined by its current activity.
- Each process may be in one of the following states:
  - **New** - The process is being created;
  - **Running** - Instructions are being executed;
  - **Waiting**

Process state

- As a process executes, it changes **state**.
  - The state of the process is defined by its current activity.
- Each process may be in one of the following states:
  - **New** - The process is being created;
  - **Running** - Instructions are being executed;
  - **Waiting** - The process is waiting for some event to occur;

Process state

- As a process executes, it changes **state**.
  - The state of the process is defined by its current activity.
- Each process may be in one of the following states:
  - **New** - The process is being created;
  - **Running** - Instructions are being executed;
  - **Waiting** - The process is waiting for some event to occur;
  - **Ready**

Process state

- As a process executes, it changes **state**.
  - The state of the process is defined by its current activity.
- Each process may be in one of the following states:
  - **New** - The process is being created;
  - **Running** - Instructions are being executed;
  - **Waiting** - The process is waiting for some event to occur;
  - **Ready** - The process is waiting to be assigned to a processor;

Process state

- As a process executes, it changes **state**.
  - The state of the process is defined by its current activity.
- Each process may be in one of the following states:
  - **New** - The process is being created;
  - **Running** - Instructions are being executed;
  - **Waiting** - The process is waiting for some event to occur;
  - **Ready** - The process is waiting to be assigned to a processor;
  - **Terminated**

Process state

- As a process executes, it changes **state**.
  - The state of the process is defined by its current activity.
- Each process may be in one of the following states:
  - **New** - The process is being created;
  - **Running** - Instructions are being executed;
  - **Waiting** - The process is waiting for some event to occur;
  - **Ready** - The process is waiting to be assigned to a processor;
  - **Terminated** - The process has finished its execution.

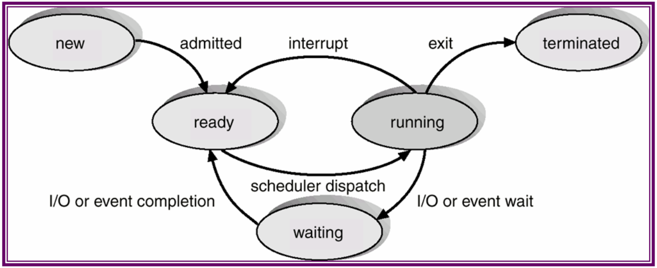
State transition (1/2)

State transition (1/2)

- As the process proceeds, it will transit from current state to another.
  - There are six transitions among these five states.

State transition (1/2)

- As the process proceeds, it will transit from current state to another.
  - There are six transitions among these five states.



State transition (2/2)

State transition (2/2)

- Transitions

State transition (2/2)

- Transitions

  Admitted The process has been created and is ready to run;

State transition (2/2)

- Transitions

  Admitted The process has been created and is ready to run;

  Scheduler dispatch The **scheduler** picks the process to run;

State transition (2/2)

- Transitions

  Admitted The process has been created and is ready to run;

  Scheduler dispatch The **scheduler** picks the process to run;

  Interrupt An interrupt has occurred;

State transition (2/2)

- Transitions

  Admitted The process has been created and is ready to run;

  Scheduler dispatch The **scheduler** picks the process to run;

  Interrupt An interrupt has occurred;

  I/O event wait The process blocks for I/O completion or reception of an signal;

State transition (2/2)

- Transitions

  Admitted: The process has been created and is ready to run;

  Scheduler dispatch: The **scheduler** picks the process to run;

  Interrupt: An interrupt has occurred;

  I/O event wait: The process blocks for I/O completion or reception of an signal;

  I/O event completion: I/O has completed or an event has occurred;

State transition (2/2)

- Transitions

  Admitted The process has been created and is ready to run;

  Scheduler dispatch The **scheduler** picks the process to run;

  Interrupt An interrupt has occurred;

  I/O event wait The process blocks for I/O completion or reception of an signal;

  I/O event completion I/O has completed or an event has occurred;

  Exit The process has finished its execution.

Process Control Block

- Each process is represented in the operating system by a **Process Control Block (PCB)**.

Process Control Block

- Each process is represented in the operating system by a **Process Control Block (PCB)**.
    - Also known as **Task Control Block**.

Process Control Block

- Each process is represented in the operating system by a **Process Control Block (PCB)**.
    - Also known as **Task Control Block**.
- Notes

Process Control Block

- Each process is represented in the operating system by a **Process Control Block (PCB)**.
  - Also known as **Task Control Block**.
- Notes
  1. **Process number** is an unique identifier of a process, also known as **PID**.

Process Control Block

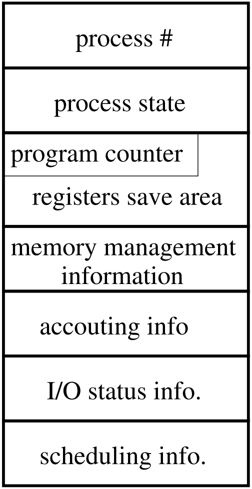
- Each process is represented in the operating system by a **Process Control Block (PCB)**.
  - Also known as **Task Control Block**.
- Notes
  1. **Process number** is an unique identifier of a process, also known as **PID**.
  2. **Program counter (PC)** is one of registers.

Process Control Block

- Each process is represented in the operating system by a **Process Control Block (PCB)**.
  - Also known as **Task Control Block**.
- Notes
  1. **Process number** is an unique identifier of a process, also known as **PID**.
  2. **Program counter (PC)** is one of registers.
  3. **Scheduling information** includes process' priority, etc.

Process Control Block

- Each process is represented in the operating system by a **Process Control Block (PCB)**.
    - Also known as **Task Control Block**.
- Notes
    1. **Process number** is an unique identifier of a process, also known as **PID**.
    2. **Program counter (PC)** is one of registers.
    3. **Scheduling information** includes process' priority, etc.



Questions

- Any questions?

Outline

Process scheduling

Process scheduling

- When running process can not proceed for some reason, the operating system must **decide which process to run next**.

Process scheduling

- When running process can not proceed for some reason, the operating system must **decide which process to run next**.
  - Three main concerns:

Process scheduling

- When running process can not proceed for some reason, the operating system must **decide which process to run next**.
  - Three main concerns:
    1. What happens to the process currently running?

Process scheduling

- When running process can not proceed for some reason, the operating system must **decide which process to run next**.
  - Three main concerns:
    1. What happens to the process currently running?
    2. How to keep track of what each process should be doing?

- When running process can not proceed for some reason, the operating system must **decide which process to run next**.
    - Three main concerns:
        1. What happens to the process currently running?
        2. How to keep track of what each process should be doing?
        3. How to decide which process to run next?

Concern 1

Concern 1

- What happens to the process currently running?

Concern 1

- What happens to the process currently running?
  - **Context switch**

Concern 1

- What happens to the process currently running?
  - **Context switch**

process #
process state
program counter registers save area
memory management information
accouting info
I/O status info.
scheduling info.

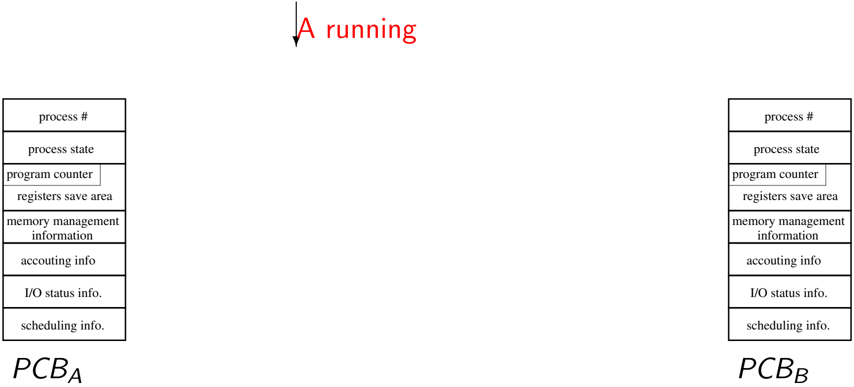
PCB_A

process #
process state
program counter registers save area
memory management information
accouting info
I/O status info.
scheduling info.

PCB_B

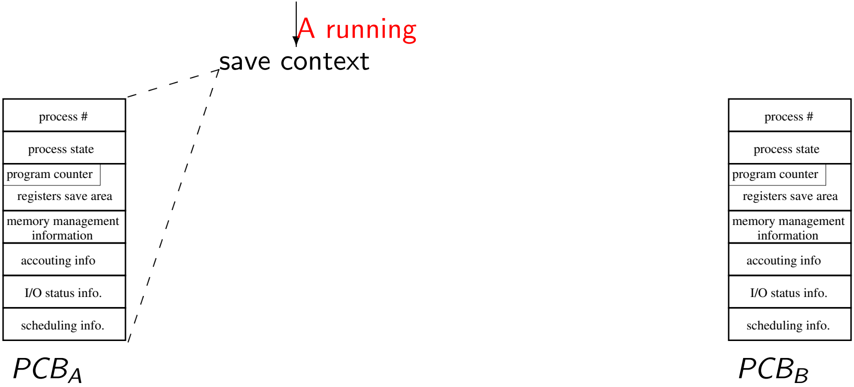
Concern 1

- What happens to the process currently running?
  - **Context switch**



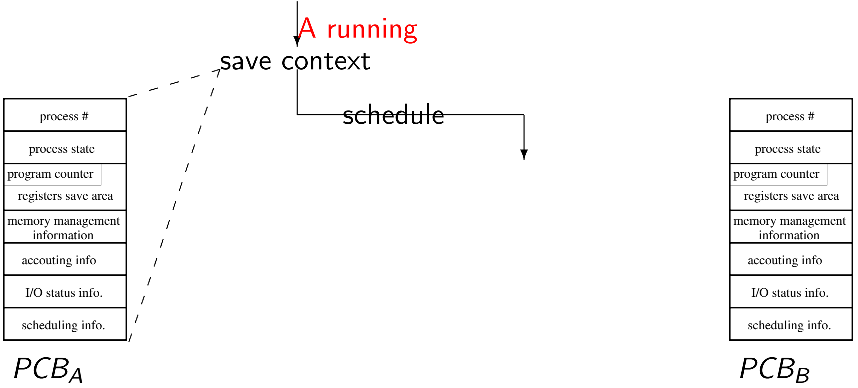
Concern 1

- What happens to the process currently running?
  - **Context switch**



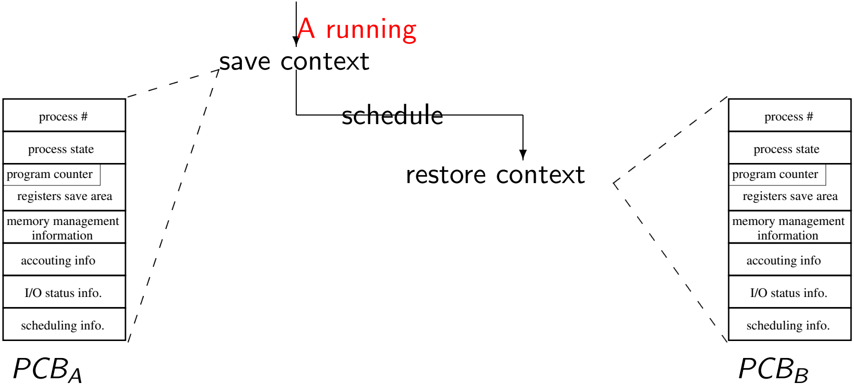
Concern 1

- What happens to the process currently running?
  - **Context switch**

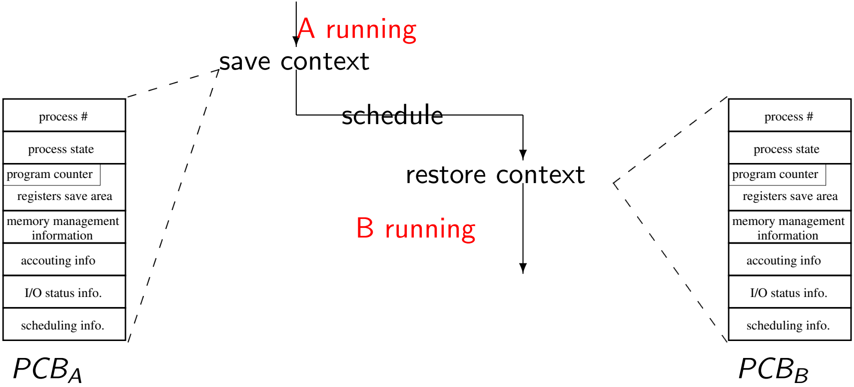


Concern 1

- What happens to the process currently running?
  - **Context switch**

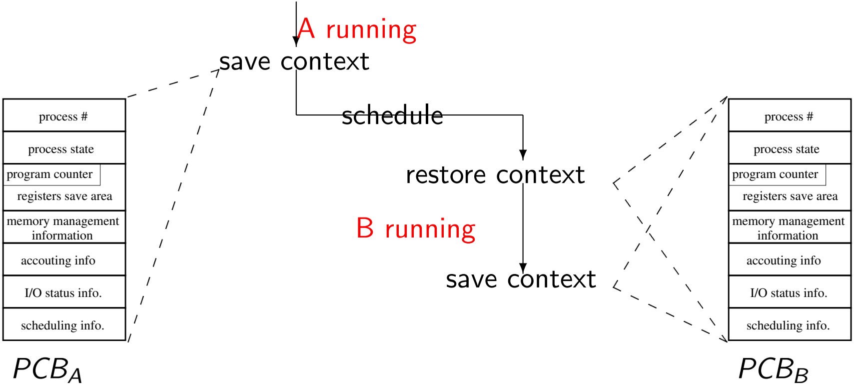


- What happens to the process currently running?
  - **Context switch**



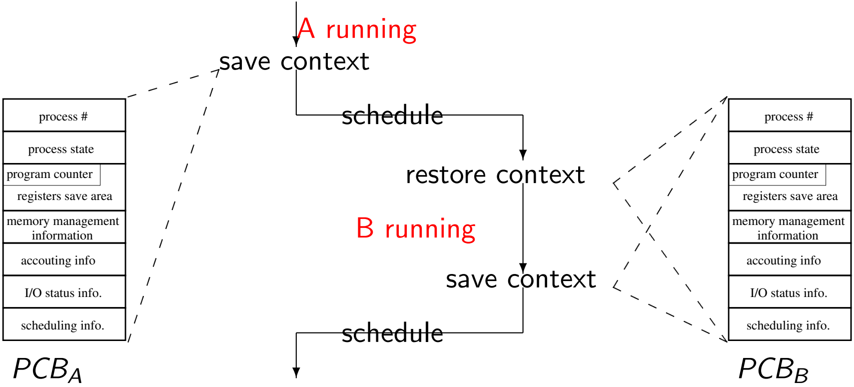
Concern 1

- What happens to the process currently running?
  - **Context switch**



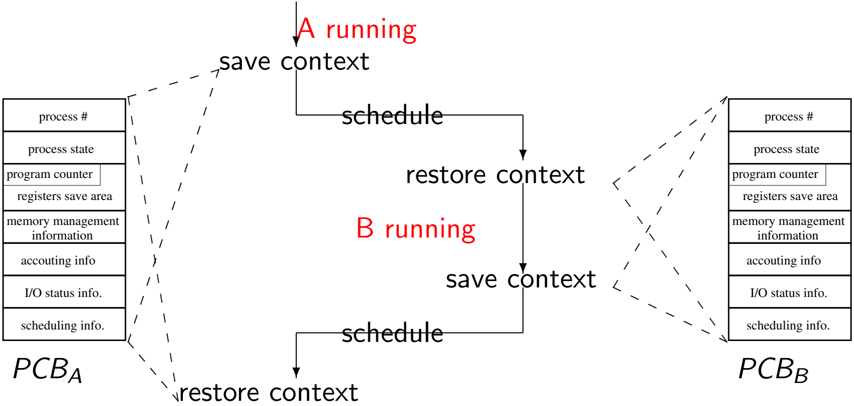
Concern 1

- What happens to the process currently running?
  - **Context switch**



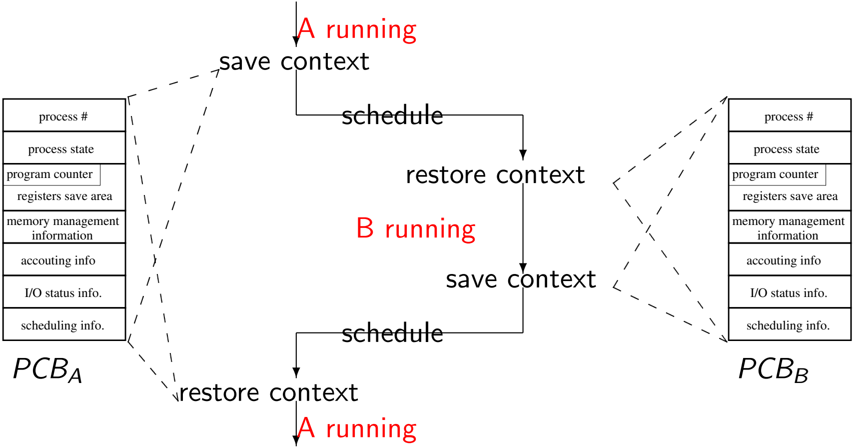
Concern 1

- What happens to the process currently running?
  - **Context switch**



Concern 1

- What happens to the process currently running?
  - **Context switch**



Context switch

Context switch

- Context switch is **pure** overhead.

Context switch

- Context switch is **pure** overhead.
  - Depending on the underling processor, it may burn **a lot of** CPU cycles.

Context switch

- Context switch is **pure** overhead.
  - Depending on the underling processor, it may burn **a lot of** CPU cycles.
  - Context switching has become such as performance bottleneck that programmers are using a new structure to avoid it whenever possible.

Concern 2

Concern 2

- How to keep track of what each process should be doing?

Concern 2

- How to keep track of what each process should be doing?
    - **Scheduling queues**

Concern 2

- How to keep track of what each process should be doing?
  - **Scheduling queues**
    - **Job queue**

Concern 2

- How to keep track of what each process should be doing?
  - **Scheduling queues**
    - **Job queue** - consists of all processes in the system;

Concern 2

- How to keep track of what each process should be doing?
  - **Scheduling queues**
    - **Job queue** - consists of all processes in the system;
    - **Ready queue**

Concern 2

- How to keep track of what each process should be doing?
  - **Scheduling queues**
    - **Job queue** - consists of all processes in the system;
    - **Ready queue** - consists of processes waiting for CPU to execute;

Concern 2

- How to keep track of what each process should be doing?
  - **Scheduling queues**
    - **Job queue** - consists of all processes in the system;
    - **Ready queue** - consists of processes waiting for CPU to execute;
  - The operating system also has other queues.

Concern 2

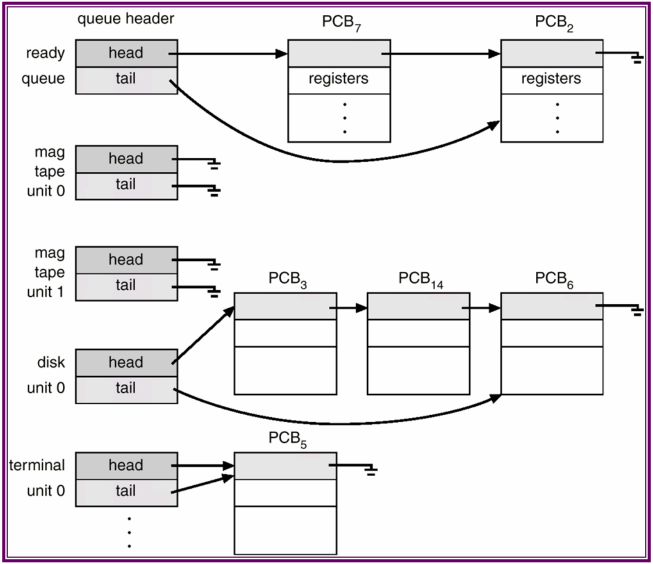
- How to keep track of what each process should be doing?
  - **Scheduling queues**
    - **Job queue** - consists of all processes in the system;
    - **Ready queue** - consists of processes waiting for CPU to execute;
  - The operating system also has other queues.
    - **I/O device queue**

Concern 2

- How to keep track of what each process should be doing?
  - **Scheduling queues**
    - **Job queue** - consists of all processes in the system;
    - **Ready queue** - consists of processes waiting for CPU to execute;
  - The operating system also has other queues.
    - **I/O device queue** - consists of processes waiting for a particular I/O device.

Various I/O device queues

Various I/O device queues



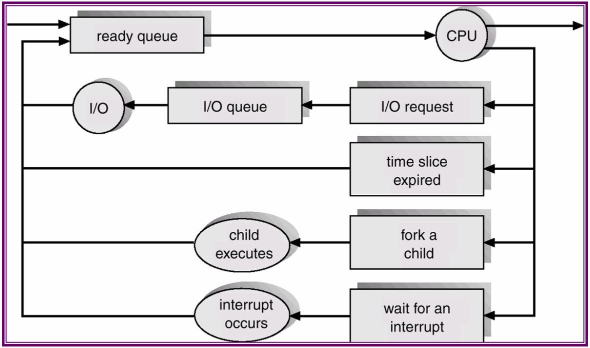
Queuing diagram

Queuing diagram

- The queuing diagram is a common representation of process scheduling in the operating system.

Queuing diagram

- The queuing diagram is a common representation of process scheduling in the operating system.



Concern 3

Concern 3

- How to decide which process to run next?

Concern 3

- How to decide which process to run next?
    - It's the **scheduler** which makes this decision.

Concern 3

- How to decide which process to run next?
  - It's the **scheduler** which makes this decision.
- The scheduler selects a process from the ready queue and allocates CPU to it.

Concern 3

- How to decide which process to run next?
  - It's the **scheduler** which makes this decision.
- The scheduler selects a process from the ready queue and allocates CPU to it.
  - We call this scheduler the **CPU scheduler**.

Questions

- Any questions?

Outline

Operations on process (1/2)

Operations on process (1/2)

- Creation

Operations on process (1/2)

- Creation - a “parent” process spawns a “child” process.

Operations on process (1/2)

- Creation - a “parent” process spawns a “child” process.
  - Parent process create children processes, which, in turn create other processes, forming a tree of processes.

Operations on process (1/2)

- Creation - a “parent” process spawns a “child” process.
  - Parent process create children processes, which, in turn create other processes, forming a tree of processes.
  - For example, *fork* in Linux/Unix and *CreateProcess* in Windows.

Operations on process (1/2)

- Creation - a “parent” process spawns a “child” process.
  - Parent process create children processes, which, in turn create other processes, forming a tree of processes.
  - For example, *fork* in Linux/Unix and *CreateProcess* in Windows.

```
int main() {
  int pid = fork(); /*fork another process*/
  if(pid < 0) {     /*error occurred*/
    printf("fork failed\n");
    exit(-1);
  } else if (pid == 0) /*in child process*/
    execlp("/bin/ls", "ls", NULL); /*executes a program*/
  else { /*in parent process*/
    waitpid(pid,NULL,0); /*waiting for the child to exit
        */
    printf("Child exited\n");
    exit(0);
  }
}
```

Operations on process (2/2)

Operations on process (2/2)

- Termination.

Operations on process (2/2)

- Termination. Two kinds of termination:

Operations on process (2/2)

- Termination. Two kinds of termination:

  normal
  : a process asks the operating system to delete it. For example, *exit* in Linux/Unix and *ExitProcess* in Windows.

Operations on process (2/2)

- Termination. Two kinds of termination:
  - **normal** a process asks the operating system to delete it. For example, *exit* in Linux/Unix and *ExitProcess* in Windows.
  - **abnormal** one process may be killed by another process by *kill* in Linux/Unix and *TerminateProcess* in Windows.

Operations on process (2/2)

- Termination. Two kinds of termination:
  - **normal** a process asks the operating system to delete it. For example, *exit* in Linux/Unix and *ExitProcess* in Windows.
  - **abnormal** one process may be killed by another process by *kill* in Linux/Unix and *TerminateProcess* in Windows.
- Communication

Operations on process (2/2)

- Termination. Two kinds of termination:

  normal a process asks the operating system to delete it. For example, *exit* in Linux/Unix and *ExitProcess* in Windows.

  abnormal one process may be killed by another process by *kill* in Linux/Unix and *TerminateProcess* in Windows.

- Communication - facilities used by one process to communicate with another.

Operations on process (2/2)

- Termination. Two kinds of termination:

  normal a process asks the operating system to delete it. For example, *exit* in Linux/Unix and *ExitProcess* in Windows.

  abnormal one process may be killed by another process by *kill* in Linux/Unix and *TerminateProcess* in Windows.

- Communication - facilities used by one process to communicate with another.
- Cooperation

Operations on process (2/2)

- Termination. Two kinds of termination:
  - **normal** a process asks the operating system to delete it. For example, *exit* in Linux/Unix and *ExitProcess* in Windows.
  - **abnormal** one process may be killed by another process by *kill* in Linux/Unix and *TerminateProcess* in Windows.
- Communication - facilities used by one process to communicate with another.
- Cooperation - cooperating processes need to synchronize their actions.

Questions

- Any questions?

Outline

Cooperating processes

Cooperating processes

- A process can be

Cooperating processes

- A process can be
  - **independent** if it cannot affect or be affected by the other processes;

Cooperating processes

- A process can be
  - **independent** if it cannot affect or be affected by the other processes;
  - **cooperating** otherwise.

Cooperating processes

- A process can be
  - **independent** if it cannot affect or be affected by the other processes;
  - **cooperating** otherwise.
- There are several advantages for the process cooperation.

Cooperating processes

- A process can be
  - **independent** if it cannot affect or be affected by the other processes;
  - **cooperating** otherwise.
- There are several advantages for the process cooperation.

  Information sharing: Several processes may be interested in the same piece of information;

Cooperating processes

- A process can be
  - **independent** if it cannot affect or be affected by the other processes;
  - **cooperating** otherwise.
- There are several advantages for the process cooperation.

  Information sharing: Several processes may be interested in the same piece of information;

  Computation speedup: Break the problem into several subtasks that can be run in parallel;

Cooperating processes

- A process can be
  - **independent** if it cannot affect or be affected by the other processes;
  - **cooperating** otherwise.
- There are several advantages for the process cooperation.

  Information sharing: Several processes may be interested in the same piece of information;

  Computation speedup: Break the problem into several subtasks that can be run in parallel;

  Modularity: Separate processes for different functions by design;

Cooperating processes

- A process can be
  - **independent** if it cannot affect or be affected by the other processes;
  - **cooperating** otherwise.
- There are several advantages for the process cooperation.

  Information sharing: Several processes may be interested in the same piece of information;

  Computation speedup: Break the problem into several subtasks that can be run in parallel;

  Modularity: Separate processes for different functions by design;
- However, concurrent execution of cooperating processes requires mechanisms that allow processes to communicated with one another and to synchronize their actions.

Example

Example

- **The producer and consumer problem**

Example

- **The producer and consumer problem**
  - A **producer** process produces information that is consumed by a **consumer** process.

Example

- **The producer and consumer problem**
  - A **producer** process produces information that is consumed by a **consumer** process.
  - This problem is a common paradigm for cooperating system.

Example

- **The producer and consumer problem**
  - A **producer** process produces information that is consumed by a **consumer** process.
  - This problem is a common paradigm for cooperating system.
- To allow producer and consumer processes to run concurrently, we need

Example

- **The producer and consumer problem**
  - A **producer** process produces information that is consumed by a **consumer** process.
  - This problem is a common paradigm for cooperating system.
- To allow producer and consumer processes to run concurrently, we need
  - a buffer which can be filled by the producer and emptied by the consumer;

Example

- **The producer and consumer problem**
  - A **producer** process produces information that is consumed by a **consumer** process.
  - This problem is a common paradigm for cooperating system.
- To allow producer and consumer processes to run concurrently, we need
  - a buffer which can be filled by the producer and emptied by the consumer;
  - to synchronize the producer and consumer.

Example

- **The producer and consumer problem**
  - A **producer** process produces information that is consumed by a **consumer** process.
  - This problem is a common paradigm for cooperating system.
- To allow producer and consumer processes to run concurrently, we need
  - a buffer which can be filled by the producer and emptied by the consumer;
  - to synchronize the producer and consumer.
- According the size of the buffer, we have

Example

- **The producer and consumer problem**
  - A **producer** process produces information that is consumed by a **consumer** process.
  - This problem is a common paradigm for cooperating system.
- To allow producer and consumer processes to run concurrently, we need
  - a buffer which can be filled by the producer and emptied by the consumer;
  - to synchronize the producer and consumer.
- According the size of the buffer, we have
  - **unbounded-buffer** producer and consumer problem;

Example

- **The producer and consumer problem**
  - A **producer** process produces information that is consumed by a **consumer** process.
  - This problem is a common paradigm for cooperating system.
- To allow producer and consumer processes to run concurrently, we need
  - a buffer which can be filled by the producer and emptied by the consumer;
  - to synchronize the producer and consumer.
- According the size of the buffer, we have
  - **unbounded-buffer** producer and consumer problem;
  - **bounded-buffer** producer and consumer problem.

The bounded-buffer producer and consumer problem

The bounded-buffer producer and consumer problem

- The buffer can be provided by the operating system through the use of an **interprocess communication (IPC)** facility.

The bounded-buffer producer and consumer problem

- The buffer can be provided by the operating system through the use of an **interprocess communication (IPC)** facility.
  - The programmers just use the system calls to fill or empty the buffer.

The bounded-buffer producer and consumer problem

- The buffer can be provided by the operating system through the use of an **interprocess communication (IPC)** facility.
  - The programmers just use the system calls to fill or empty the buffer.
- The buffer can also be in a piece of shared memory which is accessible to both the producer and consumer.

The bounded-buffer producer and consumer problem

- The buffer can be provided by the operating system through the use of an **interprocess communication (IPC)** facility.
  - The programmers just use the system calls to fill or empty the buffer.
- The buffer can also be in a piece of shared memory which is accessible to both the producer and consumer.
  - In this case, the programmer must manage the shared buffer themselves.

Implementation

Implementation

- A solution to **shared-memory bounded-buffer producer and consumer problem**.

Implementation

- A solution to **shared-memory bounded-buffer producer and consumer problem**.

```
/* Shared variables */
#define BSIZE 10
struct {
    ....
} item buffer[BSIZE];
int in = 0, out = 0;
/*----------------------------------------------------------------------*/
/*The producer loop*/
while(1) {
  /*produce an item*/
  while(((in + 1) % BSIZE) == out)
      /*do nothing*/;
  buffer[in] = itemProduced;
  in = (in + 1) % BSIZE;
}
```

```
/*The consumer loop*/
while(1) {
  while(in == out)
     /*do nothing*/;
  itemConsumed = buffer[out];
  out = (out + 1) % BSIZE;
  /*consume the item*/
}
```

Questions

- Any questions?

Outline

Interprocess communication (IPC)

Interprocess communication (IPC)

- In the producer and consumer problem, we say that the buffer can be provided by the operating system.

Interprocess communication (IPC)

- In the producer and consumer problem, we say that the buffer can be provided by the operating system.
  - The operating system must synchronize the access of producer and consumer to the buffer.

Interprocess communication (IPC)

- In the producer and consumer problem, we say that the buffer can be provided by the operating system.
  - The operating system must synchronize the access of producer and consumer to the buffer.
- What's the IPC?

Interprocess communication (IPC)

- In the producer and consumer problem, we say that the buffer can be provided by the operating system.
  - The operating system must synchronize the access of producer and consumer to the buffer.
- What's the IPC?
  - **IPC provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space**.

Interprocess communication (IPC)

- In the producer and consumer problem, we say that the buffer can be provided by the operating system.
  - The operating system must synchronize the access of producer and consumer to the buffer.
- What's the IPC?
  - **IPC provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space**.
- Examples

Interprocess communication (IPC)

- In the producer and consumer problem, we say that the buffer can be provided by the operating system.
  - The operating system must synchronize the access of producer and consumer to the buffer.
- What's the IPC?
  - **IPC provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space**.
- Examples
  - Message-passing, pipe, socket, etc.

Interprocess communication (IPC)

- In the producer and consumer problem, we say that the buffer can be provided by the operating system.
  - The operating system must synchronize the access of producer and consumer to the buffer.
- What's the IPC?
  - **IPC provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space**.
- Examples
  - Message-passing, pipe, socket, etc.

Pipe (1/3)

Pipe (1/3)

- **Pipe**

Pipe (1/3)

- **Pipe** - a FIFO communication channel between two processes.

Pipe (1/3)

- **Pipe** - a FIFO communication channel between two processes.
  - A pipe has two ends: one for reading from it and the other for writing to it.

Pipe (1/3)

- **Pipe** - a FIFO communication channel between two processes.
  - A pipe has two ends: one for reading from it and the other for writing to it.
- There are two kinds of pipe:

Pipe (1/3)

- **Pipe** - a FIFO communication channel between two processes.
  - A pipe has two ends: one for reading from it and the other for writing to it.
- There are two kinds of pipe:
  - Anonymous pipe

Pipe (1/3)

- **Pipe** - a FIFO communication channel between two processes.
  - A pipe has two ends: one for reading from it and the other for writing to it.
- There are two kinds of pipe:
  - Anonymous pipe
    - An unnamed,simplex (one-way) channel used to transfer data between the parent and child processes.

Pipe (1/3)

- **Pipe** - a FIFO communication channel between two processes.
  - A pipe has two ends: one for reading from it and the other for writing to it.
- There are two kinds of pipe:
  - Anonymous pipe
    - An unnamed,simplex (one-way) channel used to transfer data between the parent and child processes.
    - It exists for as long as the process is running.

Pipe (1/3)

- **Pipe** - a FIFO communication channel between two processes.
  - A pipe has two ends: one for reading from it and the other for writing to it.
- There are two kinds of pipe:
  - Anonymous pipe
    - An unnamed,simplex (one-way) channel used to transfer data between the parent and child processes.
    - It exists for as long as the process is running.
  - Named pipe

Pipe (1/3)

- **Pipe** - a FIFO communication channel between two processes.
  - A pipe has two ends: one for reading from it and the other for writing to it.
- There are two kinds of pipe:
  - Anonymous pipe
    - An unnamed,simplex (one-way) channel used to transfer data between the parent and child processes.
    - It exists for as long as the process is running.
  - Named pipe
    - As an extension of unnamed pipe, it can be simplex or full-duplex and may be used by any process to communicate with each other.

Pipe (1/3)

- **Pipe** - a FIFO communication channel between two processes.
  - A pipe has two ends: one for reading from it and the other for writing to it.
- There are two kinds of pipe:
  - Anonymous pipe
    - An unnamed,simplex (one-way) channel used to transfer data between the parent and child processes.
    - It exists for as long as the process is running.
  - Named pipe
    - As an extension of unnamed pipe, it can be simplex or full-duplex and may be used by any process to communicate with each other.
    - It can be permanent (as in Linux/Unix) or volatile (as in Windows).

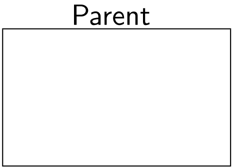
Pipe (2/3)

Pipe (2/3)

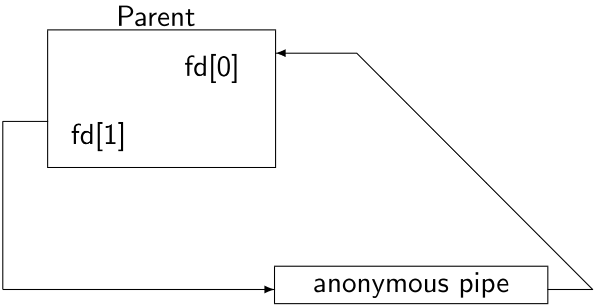
```
int main() {
    int fd[2], pid;  char buf;

    if (pipe(fd) == -1) /* Create an anonymous pipe */
    { perror("pipe"); exit(-1); }
    if ((pid = fork()) < 0)
    { perror("fork"); exit(-1); }
    if (pid == 0) {      /* Child reads from pipe */
        close(fd[1]);    /* Close unused write end */

        while (read(fd[0], &buf, 1) > 0)
            write(STDOUT_FILENO, &buf, 1);

        close(fd[0]);
        exit(0);
    } else {      /* Parent writes argv[1] to pipe */
        close(fd[0]);    /* Close unused read end */
        write(fd[1], "Hello, child!", 13);
        close(fd[1]);    /* Reader will see EOF */
        waitpid(pid,NULL,0);        /* Wait for child to
            exit*/
        exit(0);
    }
}
```

Pipe (3/3)

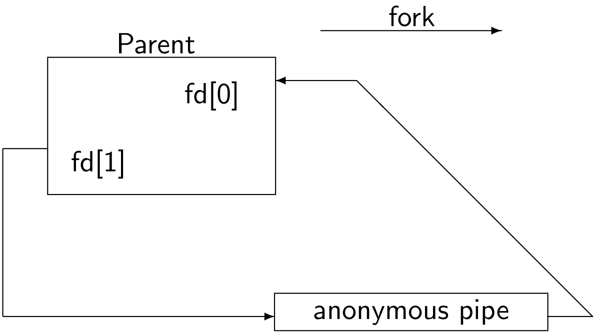
Pipe (3/3)



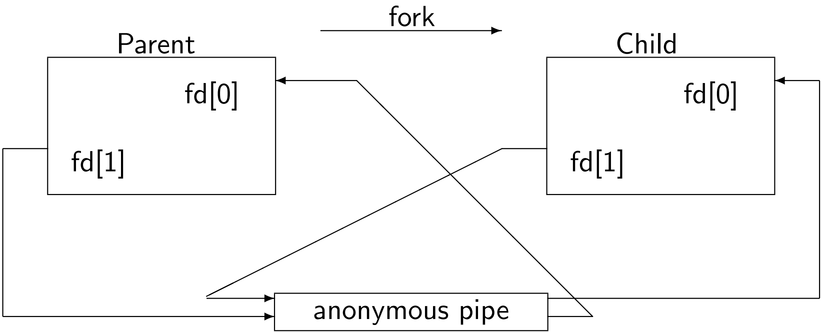
Pipe (3/3)



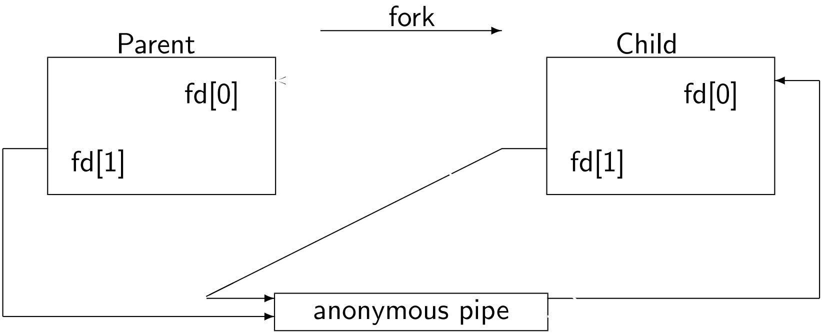
Pipe (3/3)



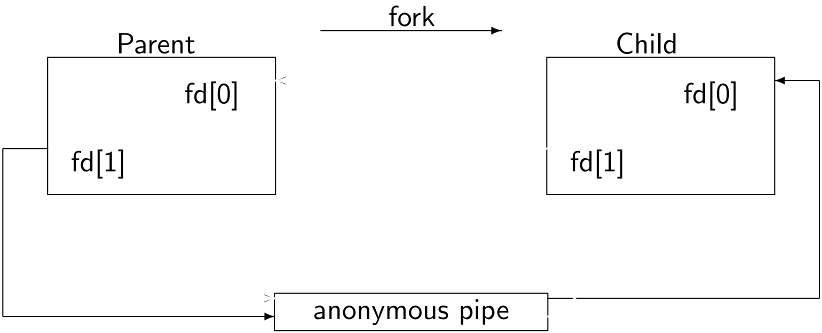
Pipe (3/3)



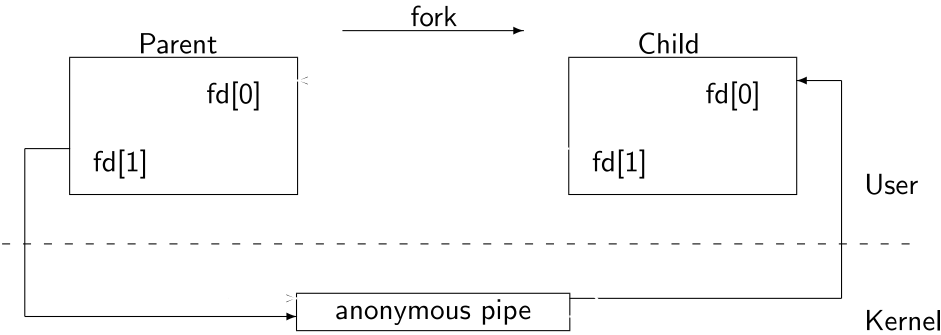
Pipe (3/3)



Pipe (3/3)



Pipe (3/3)


fork
Parent
fd[0]
fd[1]
Child
fd[0]
fd[1]
User
anonymous pipe
Kernel

Questions

- Any questions?