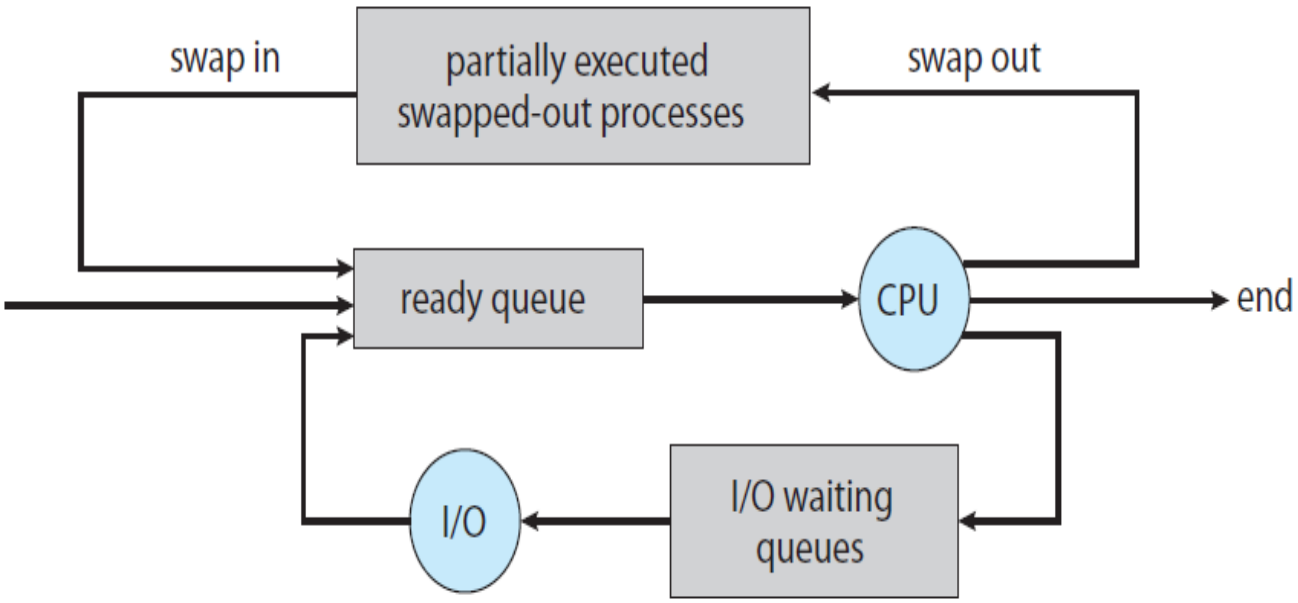

Figure 3.5 Addition of medium-term scheduling to the queueing diagram.

3.8 Context Switch

- Switching the CPU to another process requires performing **saving the state** of the current process and a **loading the saved state** of the new process. This task known as a **context switch**.
- Context-switch time is pure overhead because the system does no useful work while switching.
- The more complex the O.S the more work must be done during a context switch.

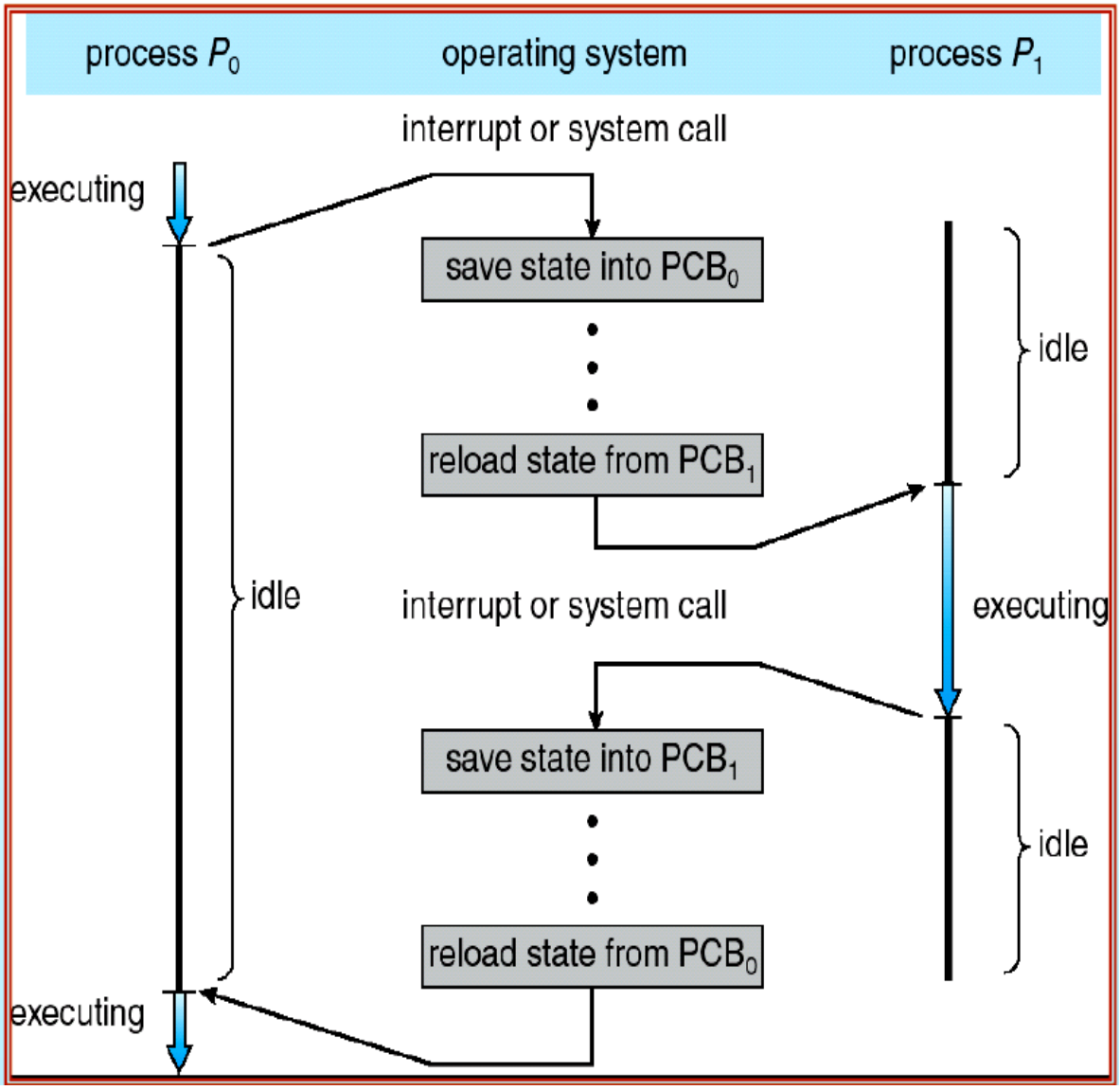

Figure 3.6 Diagram showing CPU switches from process to process.

3.9 Operations on Processes

- O.S that mange processes must be able to perform certain operation on and with processes. These include: create, destroy, suspend, resume, change a process priority, block a process, wake up a process dispatch a process, enable a process to communicate with another process.

- Creating a process involves many operations including: name a process, insert it in the ready queue, determine the process initial priority, create the PCB and allocate the process's initial resources.
- A process may create a new process. If it does the creating process is called the parent process and the created process is called the child process.

When a process creates a new process two possibilities exist in terms of execution:

a. The parent continues to execute concurrently with its children.

b. The parent waits until some or all of its children have terminated.

- A process terminates when it finishes executing its last statement and asks the O.S to delete it by using the exist system call.
- A parent may terminate the execution of one of its children for a variety at reasons such as:

a. The child has exceeded its usage of some of the resources it has been allocated.

b. The task assigned to the child is no longer required.

c. The parent is finished and the O.S does not allow a child to continue if its parent terminated.

3.10. Cooperating processes

The concurrent processes executing in the O.S may be either **independent processes** or **cooperating processes**.

A process is independent if it cannot affect or be not affected by the other processes executing in the system. Any process that does not share any data (temporary or persistent) with any other process is independent.

A process is cooperating if it can affect or be affected by the other processes executing in the system or any process that share data with other processes is a cooperating process.

There are several reasons for providing an environment that allows process cooperation:

1. Information sharing.
2. Computation speedup.
3. Modularity: Dividing the system functions into separate processes.
4. Convenience, Many tasks to work on at one time, a user may be editing, printing and compiling in parallel.

To illustrate the concept of cooperating processes let us consider the producer-consumer problem as an example of cooperating processes. A produce process produces information that is consumed by a consumer process. For example a print program produces characters that are consumed by the printer driver. To allow producer and consumer to run concurrently we must have a buffer of item that can be filled by the producer and emptied by the consumer. A producer can produce one item while the consumer is consuming another item. The producer and consumer must be synchronized. The consumer must wait until an item is produced (the buffer is empty) and the producer must wait if the buffer is full.

In the bounded-buffer and be one solution for the producer and consumer processes share the following variables:

```
var n;
type item = .....;
var buffer: array [0 .. n-l] of item;
in out: 0 .. n-1
```

With in, out initialized to the value 0. The shared buffer is implemented as a circular array with two logical pointers: in and out.
in points to the next free position in the buffer; out points to the first full position in the buffer.
The buffer is empty when in=out; the buffer is full when in+1 mod n=out.
The producer process has a local variable nextp in which the new item to be produced is stored.

```
Repeat
……
produce an item in nextp
……
while in+1 mod n=out do no-op;
buffer [in] :=nextp;
in:=in+l mod n;
until false;
The consumer process has a local variable nextc in which the item to be consumed is
stored;
repeat do-nothing instruction
while in=out do no-op;
nextc:= buffer [out],
out:=out+1 mod n;
……
consume the item in nextc; until false;
```

3.11. Thread structure

A thread sometimes called light weight process (**LWP**) is a basic unit of CPU utilization and consists of a program counter, a register set, and a stack space. It shares with peer threads its code section, data section and O.S resources such as open files and signals collectively known as a task. A traditional or heavy weight process is equal to a task with one thread.

Threads can be in one of several states ready, blocked, running, or terminated. Threads can create child threads if one thread is blocked another thread can run. Unlike processes threads are not independent of one another, because all threads can access in the task.

3.12. Interrupt Processing

An interrupt is an event that alters the sequence in which a processor executes instructions. The H/W of C/S generates the interrupt. When an interrupt occurs the following actions will be taken:

a. The O.S gains control.

b. The O.S saves the state of interrupted process in its PCB.

c. The O.S analyzes the interrupt and passes control to the appropriate routine to handle the interrupt.

d. The interrupt handler routine processes the interrupt.(IHR)

e. The state of the interrupted process (or some other next process) is restored.

f. The interrupted process (or some other next process) executes.

An interrupt may be specifically initiated by a running process (in which case it is often called a trap and said to be synchronous with the operation of the process). Or it may be caused by some event that may or may not be related to the running process. It is said to be asynchronous with the operation of the process.

3.13. Interrupt Classes (types)

There are five interrupt classes. These are:

1. SVC (Supervisor call) interrupts

A running process that executes the SVC instruction such as initiates these:

- I/O request. - Obtaining more storage. - Communicating with user operator.

2. I/O interrupts

There are initiated by the I/O H/W. such as:

- An I/O operation completes.
- An I/O error occurs.
- When a device is made ready.

3. External interrupts:

These are caused by various events including: the expiration of a quantum on an interrupting clock.

- Pressing of the console's interrupt key by the operator.
- Receipt of a signal from another processor.

4. Restart interrupts:

These occur when the operator:

- Presses the console's restart bottom.
- When a restart signal processor instruction arrives from another processor on a multi-processor system.

5. Program checks interrupt:

These are caused by many problems such as:

- Divide by zero.
- Arithmetic overflow.
- Data is in the wrong format.
- Attempt to execute invalid operation code.
- Attempt to reference a memory location beyond the limits of main memory.
- Attempt to execute a privileged instruction.
- Attempt to reference a protected resource.