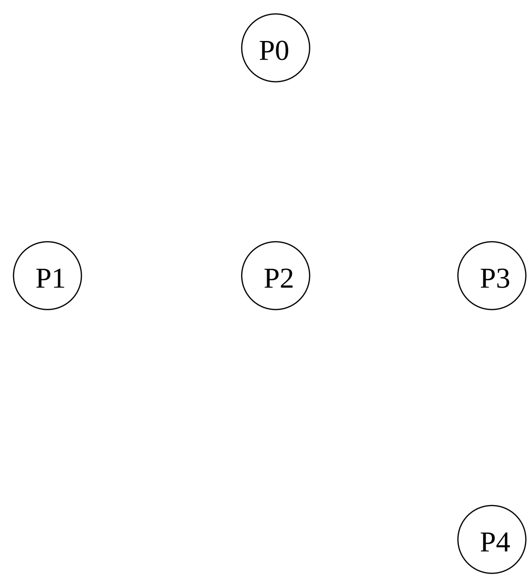
The processes in the system can execute concurrently, and they must be created and deleted dynamically. Thus the OS must provide a mechanism for process creation and termination.

A process (**parent**) may create several new processes (**children**).

For example, in Linux:



P0 creates P1,P2, and P3, and then P3 in turn creates P4.

The descriptor of each process P includes the following fields:

`p_opptr` points to the process that created P or to Process 1 (init) if the parent process no longer exists. For example, if a shell user starts a background process and exits the shell, the background process becomes child of init.

`p_cptr` points to the youngest child of P.

`p_ysptr` points to the youngest sibling, that is, to the process that has been created immediately after P by P's parent.

`p_osptr` points to the older sibling.

Shared-memory solution to the bounded-buffer problem

```
#define BUFFER_SIZE 10

typedef struct { ... } item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

The producer process has a local variable `nextProduced` in which the new item to be produced is stored:

```
while (1) {
        while (((in + 1) % BUFFER_SIZE) == out)
                ; /* buffer is full */
        buffer[in] = nextProduced;
        in = (in + 1) % BUFFER_SIZE;
}
```

The consumer process has a local variable `nextConsumed` in which the item to be consumed is stored:

```
while (1) {
        while (in == out)
                ; /* buffer is empty */
        nextConsumed = buffer[out];
        out = (out + 1) % BUFFER_SIZE;
```