Processes

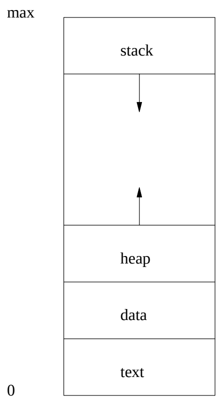
- Silberschatz chapter 3

Process concepts

- definition of a process (sometimes called a Task)
  - a program in execution
- an executable file + some kind of dynamic data
- managed by the OS

Process in Memory

-



Process Creation

- parent processes create child processes
  - which, in turn can create other processes
- forming a tree of processes

Unix shell process concepts

- a program or command typed by a user becomes a process when it is run
- the shell is a process
- shell - parent
- program or command - child
- parent may wait or continue after child has started running - use of `&`

GNU/Linux example

-
  ```
  $ gedit foo.txt &
  ```

Unix shell process commands

- `ps` process status
- `top` display and update sorted information about processes
- `pstree` display tree of processes
- `fg`, `bg`, `jobs` manage foreground, background processes
- append `&` to create a process and let the parent continue

Unix daemon concept

- a background process which executes without an associated terminal or login shell
- waits for some event or some specified task on a periodic basis
- MS Windows equivalents are called services

GNU/Linux Daemon examples

- `httpd` web server (Apache)
- `ntpd` network time protocol service
- `syslogd` system logging
- `yppasswdd`

MS Windows Services

- see the Windows Task Manager
- open up a terminal (cmd) and type the following examples:
- to list all the processes

-
  ```
  TASKLIST /svc
  ```
- to list the processes instantiated from the executable `svchost.exe`

-
  ```
  TASKLIST /FI "imagename eq svchost.exe" /svc
  ```

`to list the processes running now:`

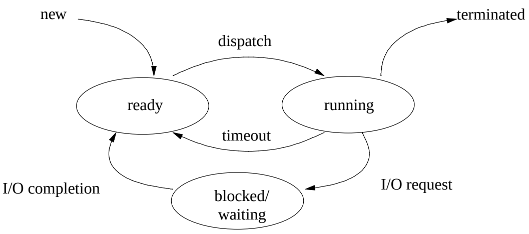
MS Windows Services

-
  ```
  TASKLIST /v /fi "STATUS eq running"
  ```

Process States

- as a process executes, it changes state:
- `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

3 State Process Diagram

- 

Self check question

- how could you modify the 3 state process diagram to incorporate partially executed swapped-out processes?

Process queues

- `Job queue`: set of all processes in the system
- `Ready queue`: set of all processes residing in main memory, ready to execute
- `Device queues`: set of processes waiting for an I/O device
- processes migrate between the various queues

Context switches

- when CPU switches to another process,
  - the system must save the state of the old process
  - and load the saved state for the new process
- the context switch time is overhead;
  - the system does no useful work while switching
- the information switched from process to process is sometimes called the volatile environment
  - or the register set of the CPU

How a Context switch is implemented in GNU Luk (comments only)

```
/*
 *  TRANSFER - save the current volatile environment into
 *             Restore the volatile environment from, p2.
 */

void SYSTEM_TRANSFER (void **p1, void *p2)
{
  /*
   *   save and disable interrupts
   *   push all registers to the stack
   *   assign p1 = top of stack
   *   assign top of stack = p2
   *   pop all registers
   *   restore interrupts
   */
}
```

- real code consists of assembly language

Unix example of Process Creation

- `fork` system call creates an exact copy of parent process
  - copied into a new address space - only difference is local pid value
- `exec` system call used to replace new process memory space with new program
- parent can wait or execute concurrently

Example fork/exec code

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main ()
{
  int pid;       /* hold process id in parent.  */
  pid = fork (); /* create new process.  */

  /* parent code - wait suspends parent until child finis
  if (pid > 0) {
    printf ("parent executing...\n");
    wait (0);
    printf ("parent finished\n");
    exit (0);
  }
  /* child code - exec date.  */
  if (pid == 0) {
    printf ("child executing...\n");
    execl ("/bin/date", "date", NULL);
    printf ("execl failed");
    exit (1);
  }
  /* should never get here.  */
  printf ("fork failed");
  exit (1);
}
```

Example fork/exec code in Python

```
import os

pid = os.fork ()
if pid > 0:
    print ’parent starting...’
    os.wait ()
    print ’parent finished’
    os._exit (0)

if pid == 0:
    print ’child starting...’
    os.system (’date’)
    os._exit (0)
    print ’Error if here’

print ’fork failed’
```