CMPSC 311

Scribe Notes Key Interfaces and Process Management (Part 1)

Lecture 13 - 2/12/16

Darshan Patel, Julian Panuci, Akash Patel, Bradley Paknejad

OVERVIEW

This lecture covers some architectural/hardware features that are essential for multiprogramming and important hardware/software interfaces of ISA, API, and ABI. Additionally, it goes into explaining the types and use of libraries. It starts a discussion of process management and covers topics like process memory layout, context, environment, states and life-cycle. The rest of process management will be covered in the next few lectures.

REVIEW OF KEY CONCEPTS

There are 3 basic concepts for the CPU

1. CPU has two different modes – User mode and Kernel Mode
    a. There are two types of instructions: **non-privileged** and **privileged**.
    b. When the CPU is in the user mode, it may only execute non-privileged instructions successfully. When it is in the kernel mode, it may execute any instruction. These modes are also called non-privileged and privileged modes, respectively.
2. **Traps** – are caused by certain events occurring inside the CPU that require the OS to run on the CPU.
3. **Interrupts** – are caused by certain events occurring outside the CPU that require the OS to run on the CPU.

INTERFACES AND ARCHITECTURE

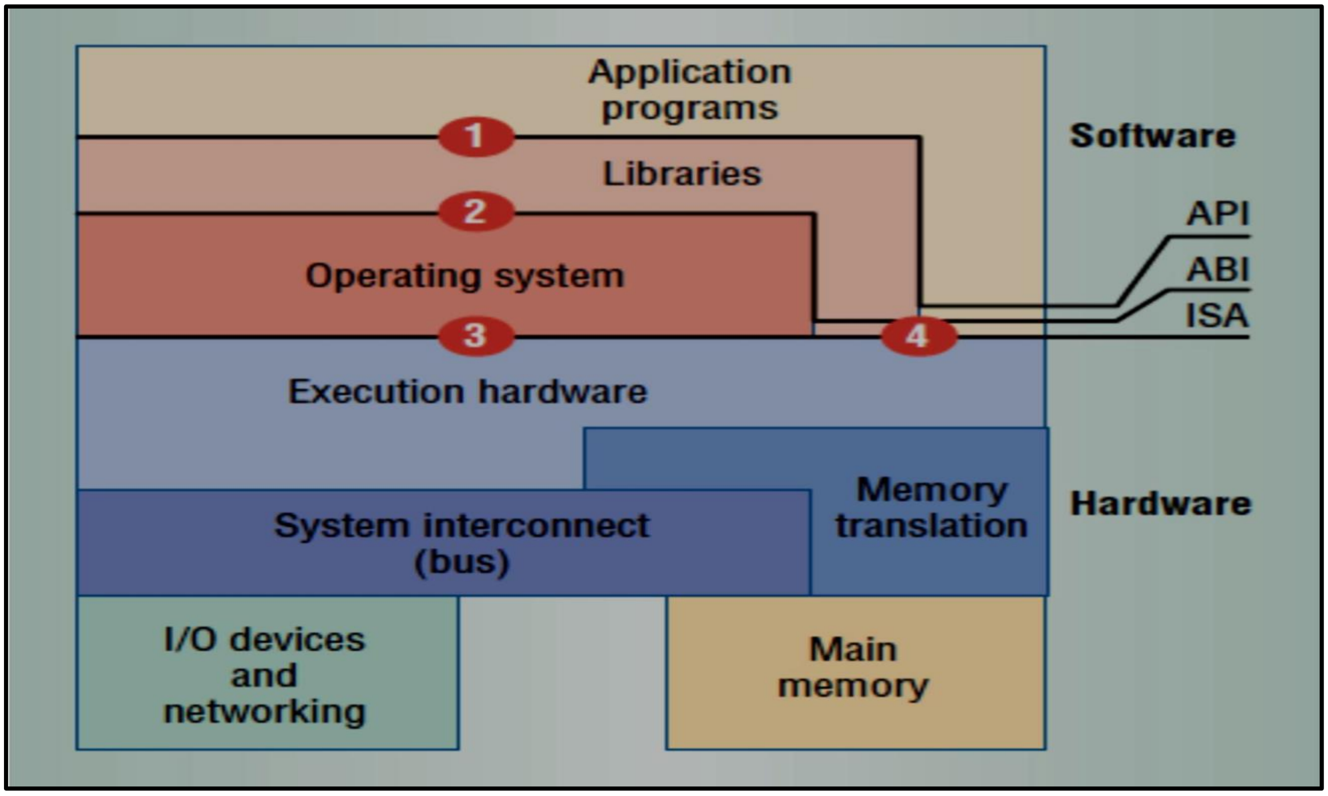


Figure 1. The above figure shows a visual abstraction of interfaces and architecture

INTERFACES

- The line separations (in bold) between the different layers of software and hardware are called interfaces.
- Figure 1 shows different interfaces such as API (Application Programming Interface), ABI (Application Binary Interface), and ISA (Instruction Set Architecture).
- The solid line in the middle of the figure (representing ISA) distinguishes the software from the hardware.
- Distinction between application programs and libraries is made for the convenience of programming.

1. Instruction Set Architecture

- The ISA is the interface between the execution hardware and software (e.g., MIPS ISA in CMPEN 331).
- This interface consists of interfaces 3 and 4. However, there is a misrepresentation in the proportion of the two interfaces. Interface 4 is supposed to be bigger than interface 3.
- Interface 4 marks the User ISA (non-privileged instructions), which is accessible to both processes and OS.
- Interface 3 makes the Privileged ISA (privileged instructions), which is only accessible to OS. This interface is responsible for managing hardware resources.

2. Application Programming Interface (API)

- API is the union of 1 and 4.

3. Application Binary Interface

- ABI is the union of 2 and 4.

In order to work with the ABI, we need to understand how system calls work and how to use them.

SYSTEM CALL

System Call Overview:

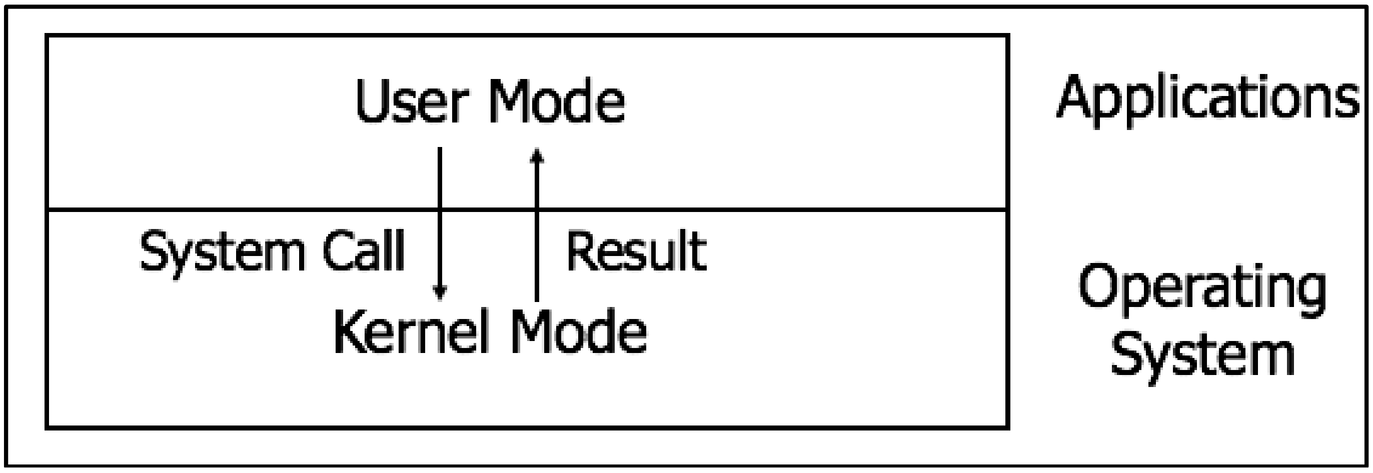


Figure 2. The figure above shows a diagram of the control flow between user and kernel modes during the lifetime of a system call.

- From the figure above, we see that applications run in user mode.
- OS runs in kernel mode.
- The system call transfers control from the application to the OS.
- The OS performs the system call and sends the result back to the application.

System Call Operation:

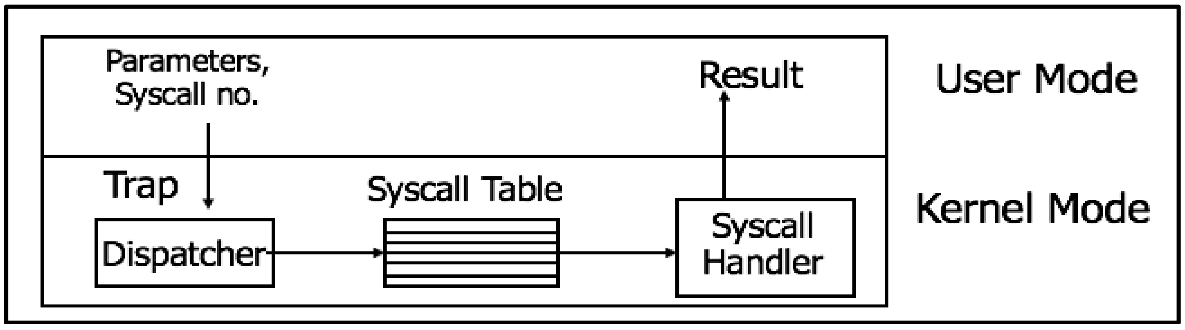


Figure 3. The figure above shows a diagram of the system call operation through individual steps in the user and kernel modes.

- From the above figure we see that the system call is made by the application in user mode. It passes the parameters for the system call. (eg: read 8 bytes from file ‘input’ into file ‘output’. The parameters for this system call would be (input, output, 8) and syscall no. would be (3)).

- As soon as the system call is made, a TRAP instruction (note that this is a deliberate trap) is executed (Note: TRAP is caused by internal sources, as mentioned earlier). Control is switched to kernel mode, and the parameters and system call no. are passed. (read instruction passes the parameters and syscall no. 3)
- Next step is the kernel looking up the system call table for the specific syscall no.
- Once the system call is found, the system call handler is invoked. (Kernel executes the code for read system call, and in effect looks up the 'input' file and transfers 8 bytes from disk to 'output' file).
- On completion of this process, system call returns the control to user mode.

Exercise: Let's consider executing a special instruction (int). This instruction is actually a trap, and thus transfers control to OS. Should this instruction be privileged or non-privileged?
(Answer: Non-privileged. However, an argument can be made for privileged as well)

System Call Implementation: This requires performing TRAP instruction, passing parameters to kernel (in registers or buffers), getting control back, and implementation in assembly language. More on this topic in the lectures to come.

LIBRARIES

- Libraries are a programmer's paradise.
- These are pre-written and pre-tested functions. Thus providing programmers with necessary tools to create meaningful programs. This saves the programmer the time of starting from scratch.
- There are two types of libraries:

I. Wrappers for system calls
   - Performs the basic function of libraries by providing convenient system calls to programmers
   - Library call does all the work behind the scenes like packaging the parameters, executing the system call, and extracting the results.
     Eg: read
   - Multiple library calls could map to same syscall
     Eg: execve -> execl, execlp, execle, execv, execvp

II. User-level utility functions
   - Implements the commonly-used functions
     Eg: Random number generation (rand, srand,…)
     String operations (strcpy, strcmp, strlen,….)

Exercise: What is the difference between wrappers for system calls and user-level utility functions?
(Answer: Wrapper for system calls is focused around system calls. User-level utilities implement functions with and without system calls)

Advantage of Libraries

- Reduce the work load on programmers
- Standard functions used across different programs eases understanding the program
- Hides complexity of functions
- The most efficient implementation is generally used for functions in libraries
- Availability for high-level languages

PROCESS MANAGEMENT

Outline Steps:

I. Process Definition
II. Process Structure and States
III. Process Creation and Execution
IV. Process Termination and Waiting

Process:

Process	Program
It's a program in execution	Consists of multiple processes
Active entity	Passive entity
Executing path of instructions	Set of instructions
Live set of resources (CPU cycles, memory)	A binary file

What does a process contain?
Program text (binary code), program counter (pointer to current instruction), data (memory required for variables, functions) and other objects like file descriptors, signals, and locks.

Process Memory Layout and Context:

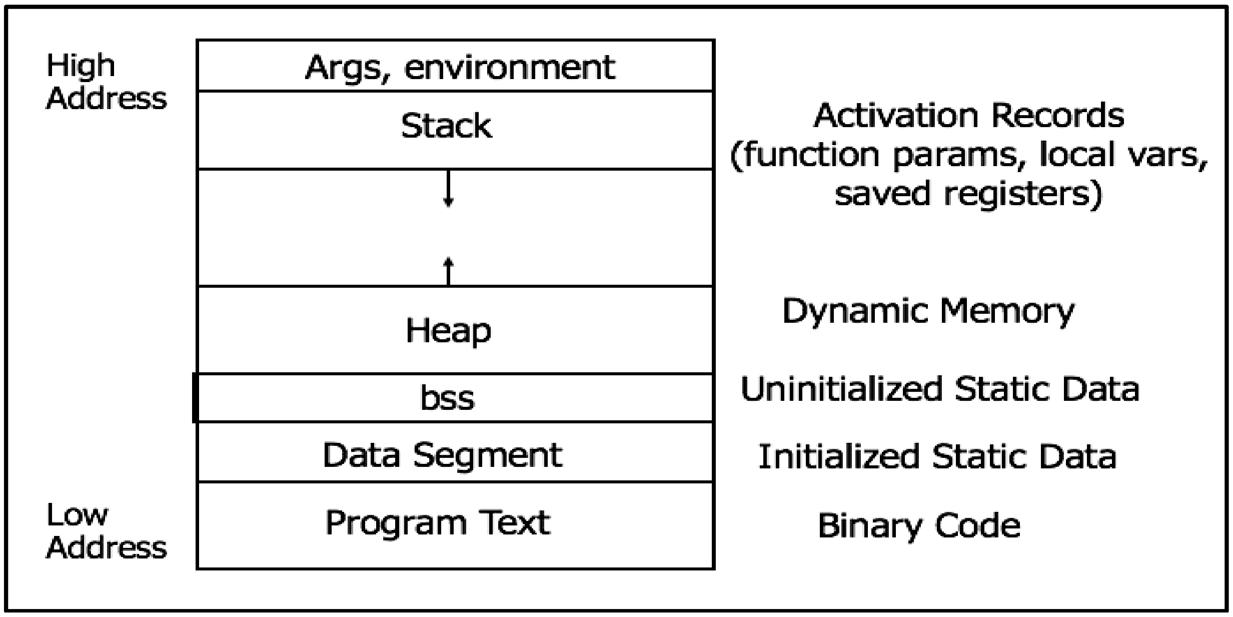


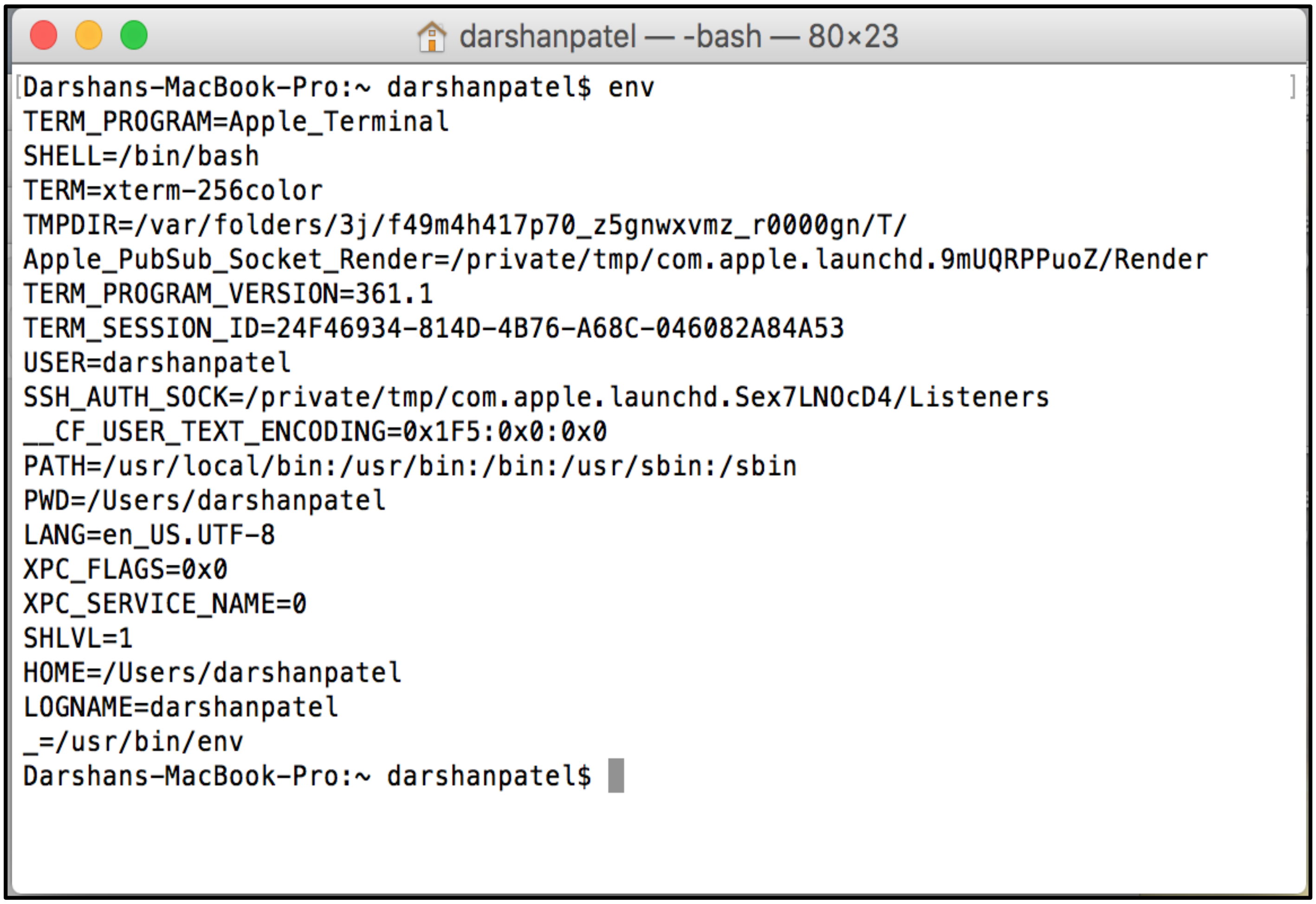
Figure 4. The figure above shows a virtual memory-mapped diagram (discussed in earlier lectures) pertaining to components accessed by a process

- The current state of process activity is maintained by a number of components shown in the figure above.
- The program counter value, the function call stack (activation records), the data values (data segment, bss, and heap), and register values determine the current state of the process context.

Exercise: What role does the OS play in the life of a process?
(Answer: OS maintains the process context)

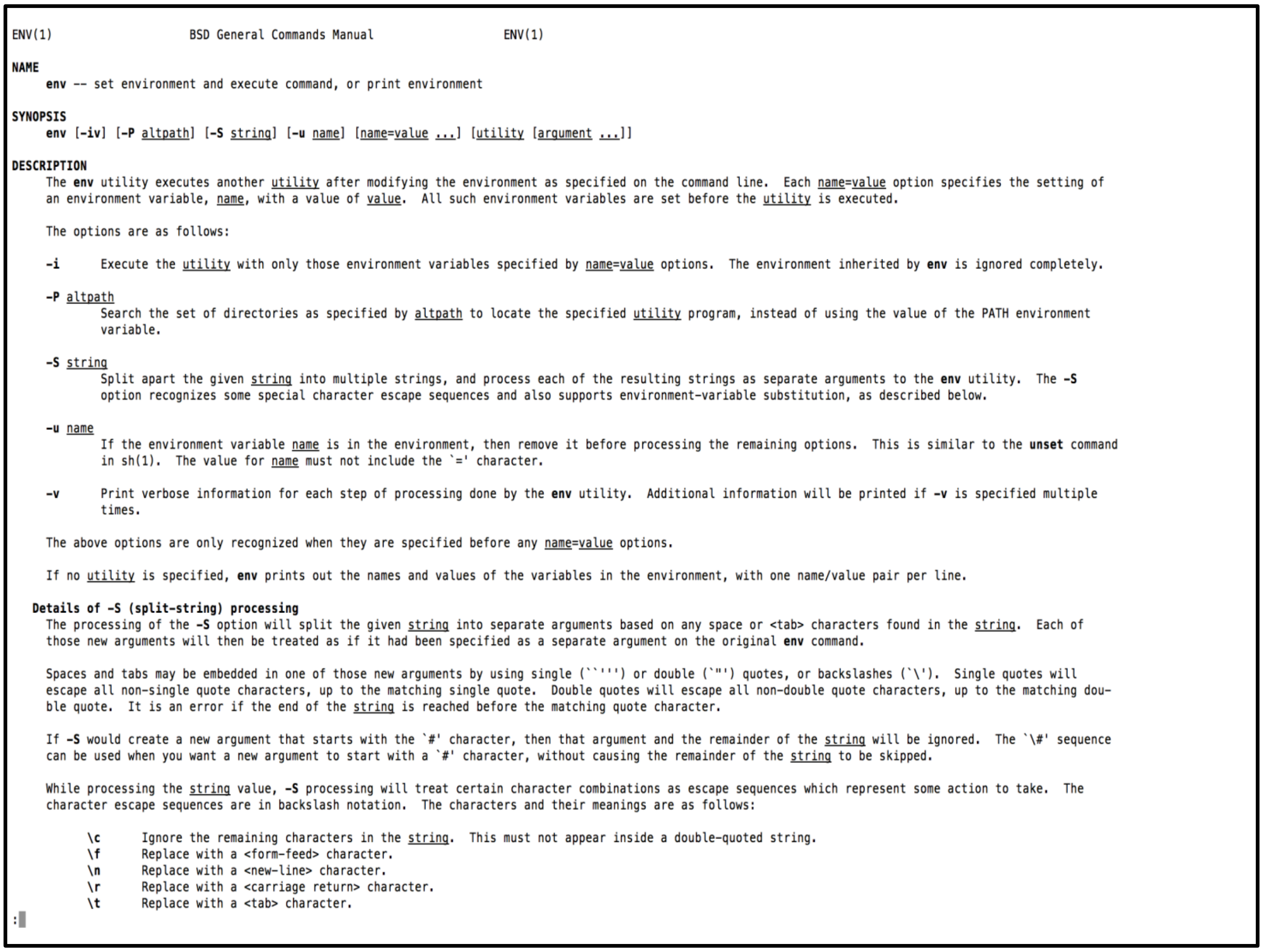
Process Environment:

- When a program is executed, the information about the context is received in one of two ways.
  1. The argv and argc arguments of the main function used to pass arguments specific to the particular program being invoked.
  2. The environment keeps track of information, that is shared by many program, changes infrequently, and is less frequently used.
- Programs executed from the shell get all of the environment variables from the shell (discussed in earlier lectures) by inheritance.
- Standard environment variables are used for information about the user's home directory, terminal type, current locale and additional variables defined (by the user) for other purposes. The set of all environment variables that have values is collectively known as the environment.
- Names of environment variables: case-sensitive; cannot contain the character '="; system defined environment variables are invariably uppercase.
- Values of environment variables: anything that can be represented as a string; must not contain an embedded null character



```
Darshans-MacBook-Pro:~ darshanpatel$ env
TERM_PROGRAM=Apple_Terminal
SHELL=/bin/bash
TERM=xterm-256color
TMPDIR=/var/folders/3j/f49m4h417p70_z5gnwxvmz_r0000gn/T/
Apple_PubSub_Socket_Render=/private/tmp/com.apple.launchd.9mUQRPPuoZ/Render
TERM_PROGRAM_VERSION=361.1
TERM_SESSION_ID=24F46934-814D-4B76-A68C-046082A84A53
USER=darshanpatel
SSH_AUTH_SOCK=/private/tmp/com.apple.launchd.Sex7LNOcD4/Listeners
__CF_USER_TEXT_ENCODING=0x1F5:0x0:0x0
PATH=/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
PWD=/Users/darshanpatel
LANG=en_US.UTF-8
XPC_FLAGS=0x0
XPC_SERVICE_NAME=0
SHLVL=1
HOME=/Users/darshanpatel
LOGNAME=darshanpatel
_=/usr/bin/env
Darshans-MacBook-Pro:~ darshanpatel$
```

Figure 5. The figure above shows the response of terminal on OSX on typing the command '$ env'. It shows details like the TERM_PROGRAM, SHELL, TMPDIR, TERM_SESSION, USER to name a few.



```
ENV(1)                  BSD General Commands Manual                  ENV(1)

NAME
     env -- set environment and execute command, or print environment

SYNOPSIS
     env [-iv] [-P altpath] [-S string] [-u name] [name=value ...] [utility [argument ...]]

DESCRIPTION
     The env utility executes another utility after modifying the environment as specified on the command line.  Each name=value option specifies the setting of
     an environment variable, name, with a value of value.  All such environment variables are set before the utility is executed.

     The options are as follows:

     -i      Execute the utility with only those environment variables specified by name=value options.  The environment inherited by env is ignored completely.

     -P altpath
             Search the set of directories as specified by altpath to locate the specified utility program, instead of using the value of the PATH environment
             variable.

     -S string
             Split apart the given string into multiple strings, and process each of the resulting strings as separate arguments to the env utility.  The -S
             option recognizes some special character escape sequences and also supports environment-variable substitution, as described below.

     -u name
             If the environment variable name is in the environment, then remove it before processing the remaining options.  This is similar to the unset command
             in sh(1).  The value for name must not include the `=' character.

     -v      Print verbose information for each step of processing done by the env utility.  Additional information will be printed if -v is specified multiple
             times.

     The above options are only recognized when they are specified before any name=value options.

     If no utility is specified, env prints out the names and values of the variables in the environment, with one name/value pair per line.

   Details of -S (split-string) processing
     The processing of the -S option will split the given string into separate arguments based on any space or <tab> characters found in the string.  Each of
     those new arguments will then be treated as if it had been specified as a separate argument on the original env command.

     Spaces and tabs may be embedded in one of those new arguments by using single (``''') or double (`"') quotes, or backslashes (`\').  Single quotes will
     escape all non-single quote characters, up to the matching single quote.  Double quotes will escape all non-double quote characters, up to the matching dou-
     ble quote.  It is an error if the end of the string is reached before the matching quote character.

     If -S would create a new argument that starts with the `#' character, then that argument and the remainder of the string will be ignored.  The `\#' sequence
     can be used when you want a new argument to start with a `#' character, without causing the remainder of the string to be skipped.

     While processing the string value, -S processing will treat certain character combinations as escape sequences which represent some action to take.  The
     character escape sequences are in backslash notation.  The characters and their meanings are as follows:

           \c      Ignore the remaining characters in the string.  This must not appear inside a double-quoted string.
           \f      Replace with a <form-feed> character.
           \n      Replace with a <new-line> character.
           \r      Replace with a <carriage return> character.
           \t      Replace with a <tab> character.
:
```

F**igure 6.** The figure above shows the definition and explanation of details of ‘env’ command (using $MAN env)

Multiprogramming:

- The key advantage of process management is that it helps multiple processes to run simultaneously.
- OS multiplexes system resources among multiple process
- Each process is allowed to run on the CPU for a quantum of time (short duration
- The process has to give up the CPU once its quantum expires or if it has to wait (timer interrupt) for an event.
- One may think of a process as a virtualized CPU.

Process States and Life Cycle:

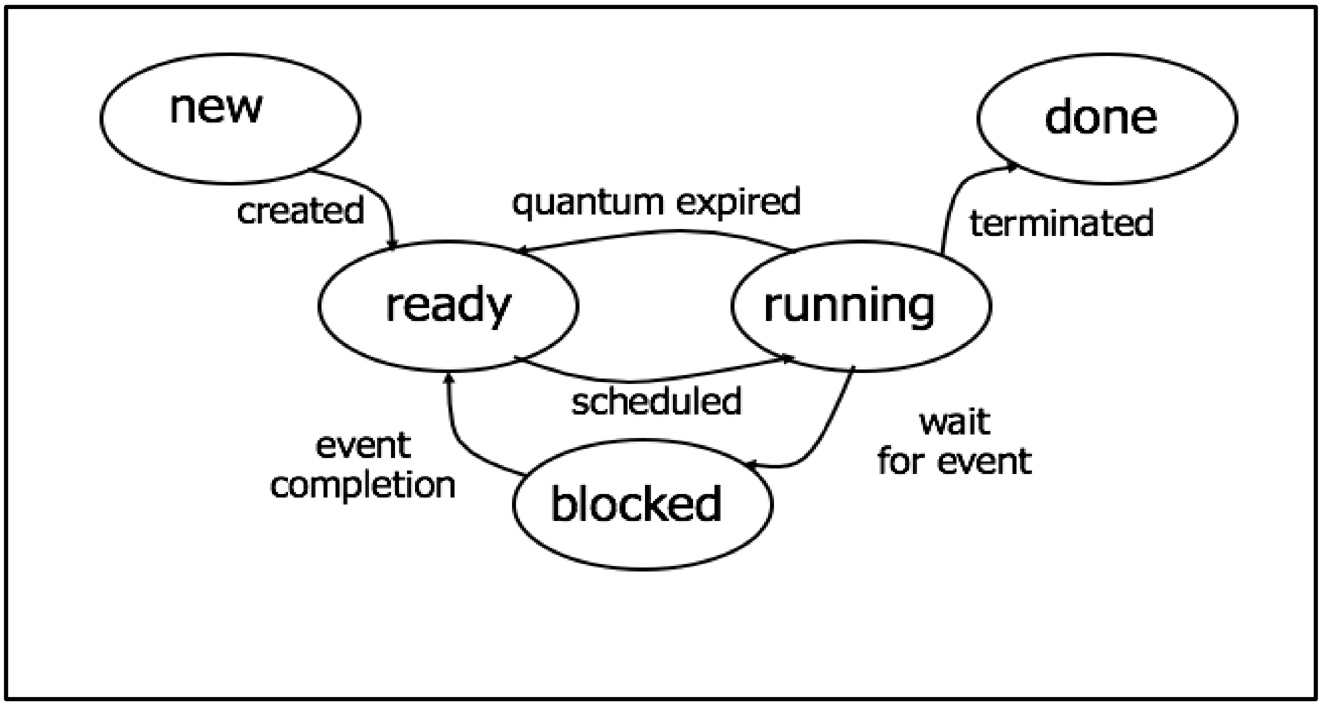


Figure 7. The figure above shows the life-cycle of a process through the different states during its lifetime.

- New process: Being created but it hasn’t started running (think of a process as a new born baby. It is aware of its birth. Similarly, a process is aware of its creation).

- Ready: Once created, it waits to be scheduled (a child might not know when it is ready to go out in the world, hence the parents nurture him/her with love, care, education and the basic necessities. Once the child is ready, it goes out into the world. The child is not aware of all the hardships the parents are going through while bring him/her up. Similarly, a process in the ready state does not realize that it is waiting to be scheduled).

- Running: On being scheduled, the programs goes into the running state.
  When the process is in the running state, it runs into one of the three possibilities
  1. The process completes running and gets terminated and goes into the done state (when a person has accomplished all that he/she wished out of life, taken care of his/her loved ones, and grown old enough in the process, he/she moves onto the last stage of life, fully aware of what’s to come. Likewise, a process is aware when it is about to be terminated).
  2. The process expires and thus it goes back to ready state, waiting to be scheduled (when a person runs into a difficulty and faces failure, they retreat from taking action, and take some time reflect on their decisions and efforts. Then they wait for the next opportunity to make a difference).
  3. The process needs to wait for an event and goes into the blocked state (when a person is waiting for a life event to happen in order to move ahead. He/she

is fully aware of this situation. Similarly, process is aware when it goes to a blocked state, in order to wait for an event).

- Blocked: The process needs to wait for an event (eg: input).
  On the completion of the event, it goes into the ready state (when a person receives an opportunity or a life event that he/she had been waiting for, and gets ready to get back to work and make a difference. In such a situation, the person is aware of the happenings around him/ her. Likewise, a process is familiar with the state it is in).

- Done: The process gets terminated (the person reaches the end of life and is on his/her way his/her way to heaven. The individual is aware of the fact that their time has come. Similarly, a process is aware when it is about to be terminated).