HY345
Assignment 1 Tutorial

04/10/2024

Assignment Description

Command Execution

Commands like:

- ls
- mkdir
- clear
- clear; ls -al

Assignment Description

Global Variables

- x=10
- my_var=”Hello world”

- echo $x *// prints: 10*
- echo $my_var *// prints: Hello world*

Assignment Description

Redirection

The **input** of a command is given from a **file instead of stdin.**

The **output** of a command is printed to a **file instead of stdout**.

- ls > y
- cat < y
- ls -al >> y

Assignment Description

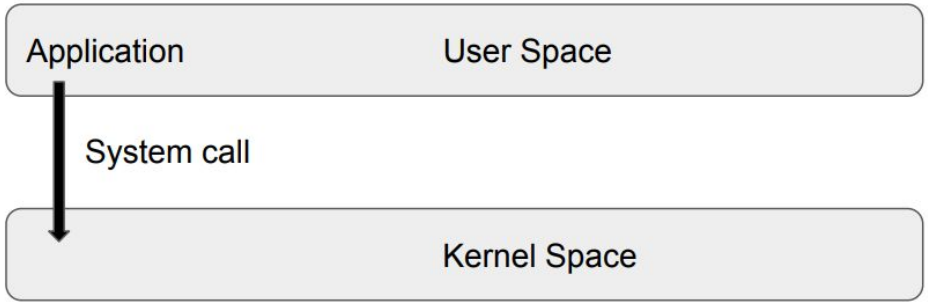
Pipes

The output of the previous command passes to the the next command as input.

- cat < y | sort | uniq | wc > y1 -> count of unique lines of y stored in y1
- ls | sort | uniq | wc -> count of unique file names in a directory

System calls

- When the user needs to call a system service a **trap instruction** needs to be called.
- The trap instruction is a procedure call that transfers control to the os.



fork()

- System call, to duplicate a process:
  - All variables have the same value in both programs
  - After execution, it returns a value in the parent and child process (pid).
    - pid<0: Fork was unsuccessful
    - pid>0: This is the parent process (The pid value is the child’s pid)
    - pid==0: This is the child process

fork()

```
while(1){               //repeat forever
    command_promt();    //display command prompt on the screen
    read_command(command,parameters);   //read input for terminal

    int pid = fork();   //fork
    if(pid == 0){       //child process,execute command

        execve(command,parameters); //execute the command

    }else if(pid > 0){  //this is the parent process

        waitpid(-1,&status,0);  //wait child process to finish its execution

    }else{

        printf("Fork not executed successfuly!\n")  //handle failed forks

    }
```

exec()

- The exec family contains multiple calls:
  - int execl(char *path, char *arg, ...);
  - int execlp(const char *file, const char *arg, ...);
  - int execle(const char *path, const char *arg, ..., char * const envp[]);
  - int execv(const char *path, char *const argv[]);
  - **int execvp(const char *file, char *const argv[]);**
  - int execvpe(const char *file, char *const argv[], char *const envp[]);

- Executes a command. **Replaces the current process with the specified program**.

wait()

- The wait() syscall **forces the parent to suspend its execution and wait for the children** process to finish its execution (or to be terminated e.g. by a signal)

- When a child process terminates, it returns an exit status to the OS, which is returned to the parent process waiting. Upon this value is received by the parent, it continues the execution

exit()

- This call **gracefully terminates the process of execution**, meaning that it cleans up and releases resources taken by this process.

- The **exit status is captured by the parent via the wait()** system call.If the parent waited and receives such signal then the child terminates (dies).If the parent wasn't waiting then the child process enters a **zombie state**.

Assignment Execution Diagram