Shell basics

Shell script starts with: #!/bin/bash and follows with commands to run

Variable, Argument and Argument Checking

Store Variables: $Var=… (no space)

- $0, $1, … – arguments passed to script ($0 is the script name)
- $@ – all arguments passed to script as array
- $# - length of the argument input. xxx.sh aaa aaa, #=2

Redirection

Command	Description	Example
>	Redirects output to a file, Creates one if it does not exist, Overrides if it does	echo hello > myfile.txt
>>	Redirects and appends output to a file	echo world >> myfile.txt
<	Redirects contents from file to stdin of another command	grep hello < myfile.txt
2>	Redirects stderr to a file	grep hi myfile.txt 2> errors.txt

Pipe → Take stdout and use this as stdin for second command

Managing Processes

fork

→ Create a *child* process which is identical to parent process, except for PID, parent process PID, resource metadata..runtime

→ Once forked, both programs continue from the point of the fork system call

Parent: fork returns PID of child

Child: fork returns 0

```
import os
import time

def main():
    pid = os.fork()

    if pid == 0:  # Child process
        print("Child process: PID =", os.getpid())
        exit(0)  # Exit child process immediately
    else:  # Parent process
        print("Parent process: PID =", os.getpid())
        print("Sleeping for 60 seconds...")
        time.sleep(60)  # Sleep for 1 minute

if __name__ == '__main__':
    main()
```

exec

→ Used to completely replace the currently running process with a different program specified

wait

→ Suspends execution of current process until one of its child processes exits

→ If child exits but parent has not called wait, child becomes a *zombie* process until wait is called

→ Stays in the process table once exited

zombie process example: PLACEHOLDER

kill

→ not really "kill" but sending signals to process, Syntax: kill <flag> <pid>

Signal	Description	Code	Example
SIGINT	Signal Interrupt	2	kill -2 22
SIGKILL	Terminate process	9	kill -9 22
SIGSTOP	Stop the process	19	kill -19 22
SIGCONT	Continue the process	18	kill -18 22

Virtual Memory

- Each process is allocated **virtual memory**
  - The process sees this as a regular, contiguous memory block
  - In reality, the OS maps different parts of this memory to the different loca RAM/mass storage in which they are stored, as in the diagram
- Why use mass storage for memory?
  - RAM has more limited size than hard drives
  - If some memory required by the program won't be accessed/changed mu makes sense to temporarily store it in mass storage
- Benefits of virtual memory
  - If process memory needs to be expanded, the page table entry for the pr can be adjusted to account for extra memory

Shell If Statement

```
if [ condition ]
then
    Do something
elif [ condition ]
then
    Do something else
else
    Do something different
fi
```

Shell While Statement

```
while program
do
    echo looping
done
```

Shell for Statement

```
for name in Alice Bob Carroll #name is a var, list after "in"
do
    echo name is $name
done
```

Memory Management

Memory for processes is stored in the following layout:

- program code instructions: at the bottom of program
- static and global data
  - data available to the whole program and constants often
- **heap - used to store data structures.**
  - eg. objects in OOP code
  - grows upwards to expand
- **Stack -** used to store temporary variables
  - e.g. variables which are used within a function, or function arguments
  - often discarded when not needed
  - grows **downwards** to expand

File System

A file system describes a way to lay out/represent file data on mass storage

Operating systems have file system module, which provides an interface to the namespace to interact with files from mass storage, or other devices

File systems can also "mount" 'files' which display information about the system

Example: /proc directory on Linux.

This means there are many implementations of file systems, optimised for different features such as speed, reliability/redundancy etc

For example: **FAT16, FAT32, exFAT** in the Windows operating system, **ext2, ext4, BtrFS, ZFS** etc for the Linux operating system.

Accessing Memory on mass storage

- What happens if a process needs to access memory on mass storage?
  - Request for access will throw an *interrupt* to OS kernel – essentially telling CPU to save what it's doing and handle this request
  - Kernel finds section on disk, and writes it to free area of RAM
    - If there is no free RAM left, the kernel must take some existing block of memory from a process in the RAM, and *swap* it to mass storage
  - Page table updated to account for this changes
- Handy features of virtual memory
  - "shared memory" – processes can have parts of their memory mapped to the same physical block
  - "copy on write" – when a process is forked, both parent and child share the same memory, until it needs to be changed
- Find info about virtual and "real" memory of a process
  - In ps:
    - the VSZ field tells us about the size of the process's virtual memory
    - the RSS field tells us about how much physical RAM a process is using

Scheduling

- Sleeping: (maybe
  - waiting for resource (e.g. file, I/O device) to become available
  - waiting to be executed by CPU
- Priority of process: indication and changing it
  - Range from negative integers (highest priority) to positive integers (lowest priority)
  - On Linux, by default, the priority for processes started by users is 20
  - nice value
    - nice value adds/subtracts a number to a process's priority to modify its default priority
    - nice value is separated from the priority itself
    - Valid nice values typically range from -20 (highest priority) to 19 (lowest priority).
    - Default value is 0 (has a neutral priority)

```
$ nice -n 5 sleep 30
$ renice 10 -p 1203
```

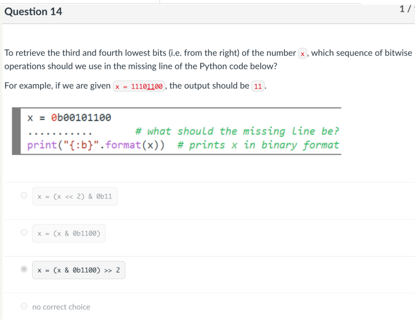
Boot Sequence

power on	hardware self-checks
BIOS running	in flash memory
MBR loaded	from 1st block of disk
GRUB loaded	also from early part of disk
kernel loaded	started by kernel from file system
init started, PID=1	started by kernel from file system
system processes running	started by init
system operating normally	/

Some Common command line & flags:

```
echo -n
# -n flag basically cancel the new line at the end of each line echoed
mkdir
rm #remove
cp #copy
touch #create files
sort #sorting -f, --ignore-case // -r, --reverse // -u, --unique
     # // -s, --stable
ls -l #view metadata with files in the working directory
chmod <group name><+/-><premission>
find "PATH" -type f/d -name ".txt"
grep -i '^[^u]*[aeio][^u]*$' /usr/share/dict/words | sort -r
#regular expression  -i case insensitive
tail #
head #
cat #reading content

cut -d "," -f 1,4 "yellow_pages.csv"
# -d is like split by ",", -f (LIST) is choosing the outcome of the cut
# -b selecting certain bytes, -c = -b but only chars
```



Question 14

1 /

To retrieve the third and fourth lowest bits (i.e. from the right) of the number `x`, which sequence of bitwise operations should we use in the missing line of the Python code below?

For example, if we are given `x = 11101100`, the output should be `11`.

```
x = 0b00101100
...........        # what should the missing line be?
print("{:b}".format(x))  # prints x in binary format
```

○ `x = (x << 2) & 0b11`

○ `x = (x & 0b1100)`

● `x = (x & 0b1100) >> 2`

○ no correct choice

Processes

- **Program**: executable sequence of instructions stored on mass storage
- **Process**: instructions of program stored in memory, and any other data required by program

unix system has many process stored in memory at same time

- **For a single CPU, only one program can run at a time**
- **UID associated to process stating who started the process, superuser UID = 0**
- **Metadata of file can be seen by running ls -l**

  type of file (- = normal file, d = directory, l = link b = block file)

  permissions (r = readable, w = writable, x = executable (for files)/accessible (for directories)) – listed as [r/-][w/-][x/-] for each group, where - means permission doesn't exist

  user permissions | group permissions | permissions for users in other groups

  - **Changing Permission using chmod command:**
- **Namespace**