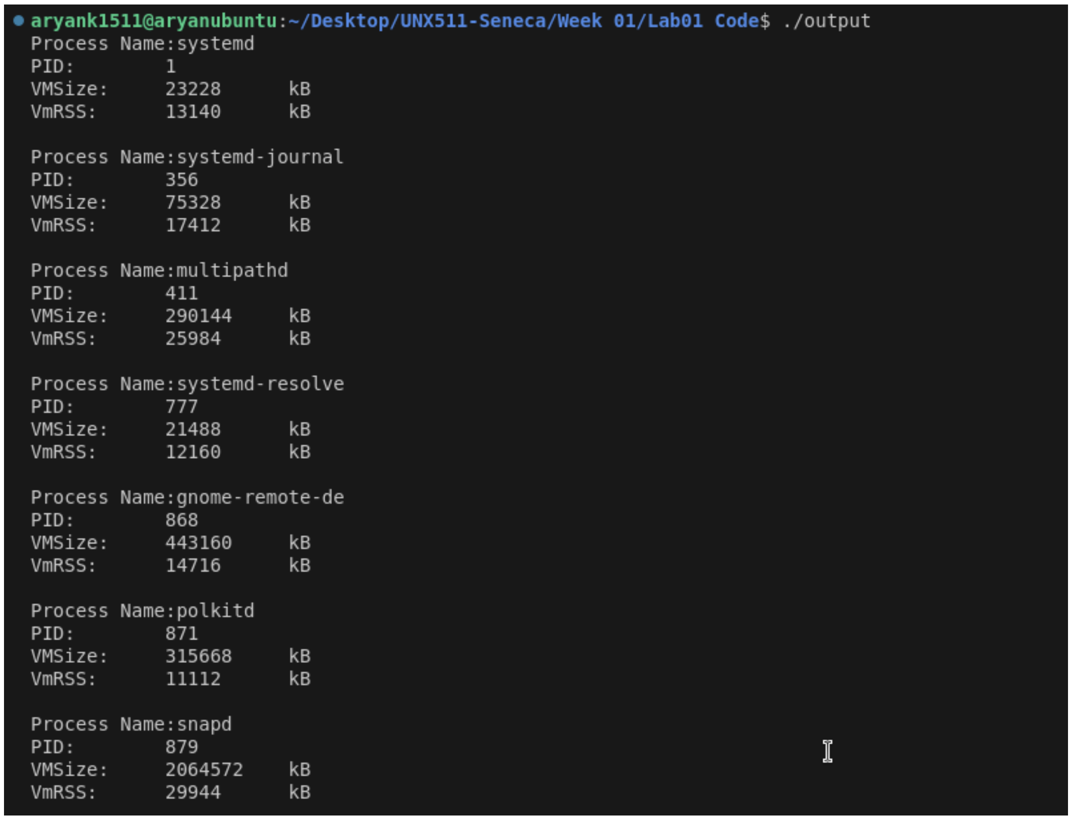
UNX511NZA Lab 2

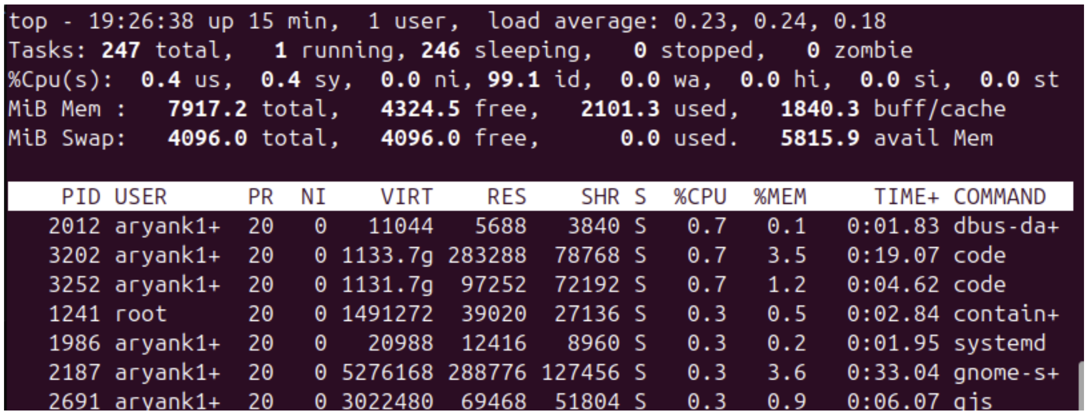
Name	Aryan Khurana
Email	akhurana22@myseneca.ca
Student ID	145282216
Professor	Mohammed Nooruddin
Date	September 18, 2024

1. Screenshot of the C++ program executing on the Ubuntu virtual machine



```
aryank1511@aryanubuntu:~/Desktop/UNX511-Seneca/Week 01/Lab01 Code$ ./output
 Process Name:systemd
 PID:        1
 VMSize:     23228      kB
 VmRSS:      13140      kB

 Process Name:systemd-journal
 PID:        356
 VMSize:     75328      kB
 VmRSS:      17412      kB

 Process Name:multipathd
 PID:        411
 VMSize:     290144     kB
 VmRSS:      25984      kB

 Process Name:systemd-resolve
 PID:        777
 VMSize:     21488      kB
 VmRSS:      12160      kB

 Process Name:gnome-remote-de
 PID:        868
 VMSize:     443160     kB
 VmRSS:      14716      kB

 Process Name:polkitd
 PID:        871
 VMSize:     315668     kB
 VmRSS:      11112      kB

 Process Name:snapd
 PID:        879
 VMSize:     2064572    kB
 VmRSS:      29944      kB
```

2. Screenshot of the top command running on the Ubuntu VM



```
top - 19:26:38 up 15 min,  1 user,  load average: 0.23, 0.24, 0.18
Tasks: 247 total,   1 running, 246 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.4 us,  0.4 sy,  0.0 ni, 99.1 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :  7917.2 total,  4324.5 free,  2101.3 used,  1840.3 buff/cache
MiB Swap:  4096.0 total,  4096.0 free,     0.0 used.  5815.9 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   2012 aryank1+  20   0   11044   5688   3840 S   0.7   0.1   0:01.83 dbus-da+
   3202 aryank1+  20   0 1133.7g 283288  78768 S   0.7   3.5   0:19.07 code
   3252 aryank1+  20   0 1131.7g  97252  72192 S   0.7   1.2   0:04.62 code
   1241 root      20   0 1491272  39020  27136 S   0.3   0.5   0:02.84 contain+
   1986 aryank1+  20   0   20988  12416   8960 S   0.3   0.2   0:01.95 systemd
   2187 aryank1+  20   0 5276168 288776 127456 S   0.3   3.6   0:33.04 gnome-s+
   2691 aryank1+  20   0 3022480  69468  51804 S   0.3   0.9   0:06.07 gjs
```

3. Difference between virtual memory and resident memory

Resident Memory: This is a measure of the RAM that a particular process is using. This is the more important number as this is what matters when you look at the memory used by a process.

Virtual Memory: This is how much memory your program thinks it is using. This is much bigger than the resident memory as a process is able to request a lot more memory than it would be using. This memory isn't allocated unless there is a need to do so. Virtual memory allows programs to run even when the system doesn't have enough physical memory, by swapping data between RAM and disk as needed.

4. A brief overview of some of the processes outputted by the program

systemd
PID: 1
VMSize: 23,228 kB
VmRSS: 13,140 kB

This is the service manager for Linux-based operating systems. This process is also known as init. It provides a system and service manager that runs as PID 1 and starts the rest of the system.

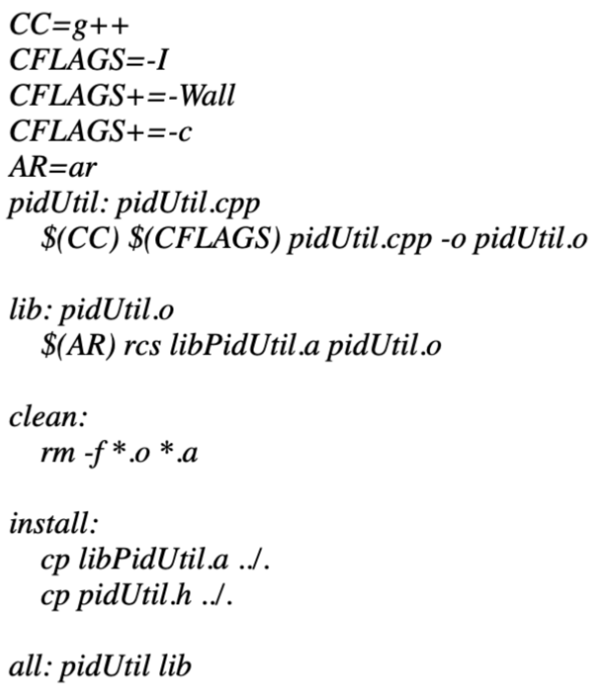
systemd-journal
PID: 356
VMSize: 75,588 kB
VmRSS: 18,692 kB

This is a component of the `systemd` suite responsible for collecting and managing system logs. It gathers log messages from various services and system components and stores them in binary log files. These logs can be used for troubleshooting, monitoring, and auditing system activities. The `systemd-journal` process is integral to the `journald` service, which handles logging and log rotation.

multipathd
PID: 411
VMSize: 290,144 kB
VmRSS: 25,984 kB

This is a daemon used in Linux environments to manage multiple paths to storage devices. It is part of the device-mapper-multipath package and helps to ensure high availability and load balancing for storage systems. By providing multiple paths to a single storage device, `multipathd` improves fault tolerance and performance, as it can reroute traffic if one path fails or become overloaded.

5. Explaining the Makefile Provided (Lab 02 Part B)



```
CC=g++
CFLAGS=-I
CFLAGS+=-Wall
CFLAGS+=-c
AR=ar
pidUtil: pidUtil.cpp
    $(CC) $(CFLAGS) pidUtil.cpp -o pidUtil.o

lib: pidUtil.o
    $(AR) rcs libPidUtil.a pidUtil.o

clean:
    rm -f *.o *.a

install:
    cp libPidUtil.a ../.
    cp pidUtil.h ../.

all: pidUtil lib
```

CC=g++

CC is just a variable name, but it's common practice to name it 'CC', which stands for C Compiler. Here, we're assigning the C++ compiler `g++` to the CC variable.

CFLAGS=-I

The '-I' flag specifies the include path for headers.

CFLAGS+=-Wall

'-Wall' stands for 'warnings all` and enables all the compiler's warning messages.

CFLAGS+=-c

The '-c' flag stands for 'compile only'. This tells the compiler to compile the source file into an object file (.o). An object file is a file that contains machine code generated by a compiler from a source file. However, it is not a complete executable yet because it hasn't been linked with other object files or libraries.

AR=ar

Here, 'ar' is the archiver tool for creating static libraries. An archiver is used to bundle multiple object files into a single static library. This static library can then be linked into other programs during the compilation process.

```
pidUtil: pidUtil.cpp
        $(CC) $(CFLAGS) pidUtil.cpp -o pidUtil.o
```

This uses the `g++` compiler (referenced by `$(CC)`) and the flags `$(CFLAGS)` to compile 'pidUtil.cpp' into an object file 'pidUtil.o'. The `-o` flag specifies the output file name, which is 'pidUtil.o'.

```
lib: pidUtil.o
        $(AR) rcs libPidUtil.a pidUtil.o
```

Creates a static library 'libPidUtil.a' from the object file 'pidUtil.o'
The `rcs` flags mean:

- **'r' :** Replace or add files to the archive
- **'c':** Create the archive if it doesn't already exist
- **'s':** Add an index to the archive, improving lookup times when linking

```
clean:
        rm -f *.o *.a
```

Cleans up object files and static libraries. The 'rm -f' command deletes all object files (*.o) and archive files (*.a) in the directory. '-f' forces the removal without asking for confirmation, even if files don't exist.

```
install:
        cp libPidUtil.a ../.
        cp pidUtil.h ../.
```

Installs the library and header file to a parent directory . The 'cp' command copies the static library 'libPidUtil.a' and the header file 'pidUtil.h' to the parent directory.

```
all: pidUtil lib
```

This runs both the `pidUtil` and `lib` targets sequentially, ensuring that the object file is compiled before the static library is created.