Wollongong of University

School of Computer Science and Software Engineering

CSCI212/MCS9212 Interacting Systems Autumn 2012

Assignment 1 (7.5 marks)

Due 11:59pm Sunday March 25, 2012

Aim

The aim of this assignment is to become familiar with Unix processes and pipelines. Additionally students will develop an understanding of memory managers.

Part 1 – A Unix Filter

The `ps(1)` command upon execution with the `-ef` options produces the following output on banshee.

```
 UID   PID  PPID  C    STIME TTY      TIME CMD
root     0     0  0   Jul 18 ?        0:00 sched
root     1     0  0   Jul 18 ?        2:11 /etc/init -
root     2     0  0   Jul 18 ?        0:00 pageout
root     3     0  0   Jul 18 ?      103:03 fsflush
root  3047     1  0   Jul 18 ?        0:00 /usr/lib/saf/sac -t 300
xyz   5876  5187  0   Jul 21 pts/17   0:00 zsh
xyz  16589 16584  0 08:28:20 pts/21   0:00 /usr/bin/bash
```

For the purposes of this assignment you are to use `ps(1)` command.

The output is broken down into columns, the first column being the owner of the process, normally a string. The second and third columns, `PID` and `PPID`, refer to the Unix process ID and parent process ID. For the purpose of this assignment we can ignore the fourth column. The fifth and sixth column are related to when the processes started and what terminal the process is attached to. The start time can appear in two different forms, the first being the time in `hh:mm:ss`, whilst the second is a date string, normally in the form of month followed by day. The `TTY` column will normally either have a `?` or a string – the format of the string's not important.

The seventh column is the execution time, representing the total CPU time consumed. It is expressed in the form of `mmm:ss`. The last field is the command followed by its arguments. Sometimes the command will have its entire path name. Sometimes the command will be the word `<defunct>` in which case there will be no entry for the `STIME`, `TTY` columns. For this assignment you should ignore process tables entries that are `<defunct>` or are encased in brackets.

As you can see the output in the process table varies; because of this, normal text processing techniques which rely on the position of certain characters/strings fail.

Using the output of the command `ps` your program should read the output produced as standard input. In other words your program is to behave as if it were part of a pipeline. Typical execution would be as follows:

```
Ps -ef | ./a.out
```

The program should do the following.

1. For each user in the process table, output the user name of the user, the number of processes that were in the process table owned by that user, the total amount of CPU time consumed by all the processes owned by that user and the process id & process name with the longest path name (determined by counting the /'s in the path name).

2. After performing the user analysis report, the total number of processes in the process table, the average cpu time consumed by all processes, the user with the most processes in the table, the user with the least processes in the table and the user, process id and process name with the longest path name (determined by counting the /'s in the path name).

Below is a piece of example output, using the `ps(1)` process table from above.

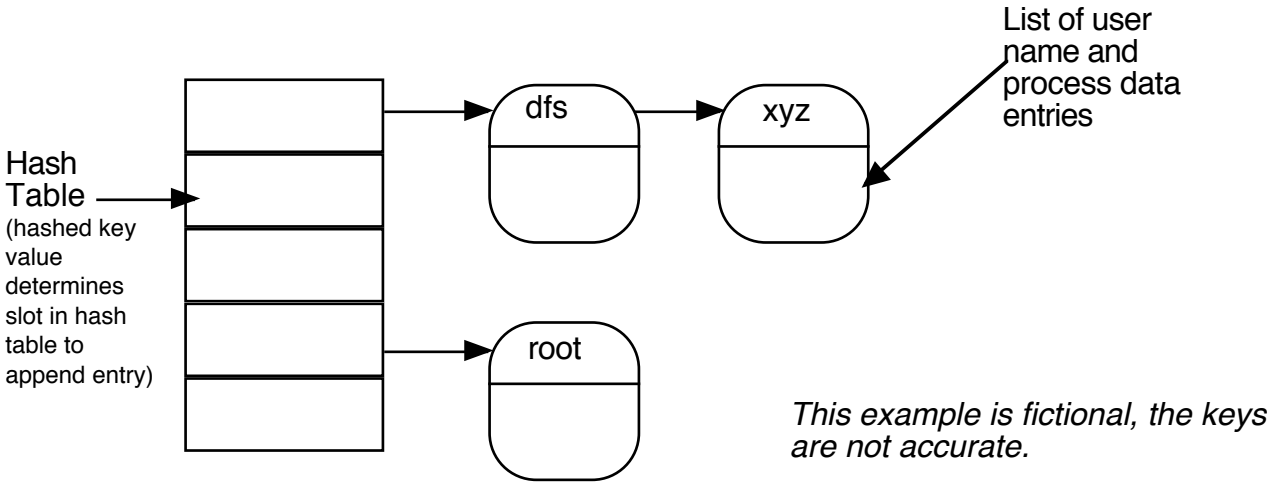
```
User root
=========

User root has a total of 5 processes
User root has consumed a total of 105 minutes and 14 seconds of CPU time.
The process id with the longest path name is pid 3047 /usr/lib/saf/sac.

User xyz
========

User xyz has a total of 2 processes
User xyz has consumed a total of 0 minutes and  second of CPU time.
The process id with the longest path name is pid 16589 /usr/bin/bash.

Statistical Summary
==================

There are a total of 7 processes in the process table.
User root has the most processes in the table (count = 5)
User xyz  has the least processes in the table (count = 2)

The process with the longest path name is pid 3047 with the path name
/usr/lib/saf/sac belonging to the root user.

The average CPU time used by all processes is 15 minutes and 2 seconds.
```

It should be noted that when the ps program is executed you will need to remove the first line which contains column headings.

In order to do this assignment, you will need to implement data structures to store this data. The underlying data structure will be a hash table using the user name as a key. The simplest function to generate the hash value for the key would be the summation of each ASCII value making up the key modulus the size of the hash table. We can expect a maximum of 100 different users logged onto the machine, so the size of your hash table should be 120.

When information is added to the hash table for a given user name, firstly the hash value is generated using the algorithm above. All information pertaining to the user and process statistics is kept in a SINGLE node. If the key does not exist in the hash table, the key and data are inserted into a list, which the hash table entry points to.

In the event of key collision, the key is to be appended to the end of a linked list. Each entry in the hash table will point to a list of entries, with each node representing a user name. A illustration is below:



In your program, you are asked to print out periods of time, all time should be printed as minutes and seconds.

Your program should check inbound data for errors i.e. it being malformed. In the event it is it should print an error and terminate.

Process

1. Design a structure for a list node, which will typically store information about processes and users.

2. Implement a linked list. The linked list should use the node you designed in step one. Put the implementation of your linked list in `linkedlist.cpp` and the header in `linkedlist.h`. *You are more then free to use classes to do this*.

3. Implement a hash table with the appropriate support routines to generate a key. The key for the hash table will be a user name. Every entry in the hash table will point to a linked list of user names. The implementation of the hash table should go in hash `table.cpp` and the header should go in hash `table.h`. *You are more then free to use classes to do this*

4. Now implement `main.cpp`, which will read `ps(1)` from standard input and perform the appropriate manipulation.. Once all input is processed, produce the reports as mentioned above.

Memory should be cleaned up prior to termination.

The compile directive will be:

```
g++ linkedlist.cpp hashtable.cpp main.cpp –o ass1
```

Submit:

Submit the source files `linkedlist.cpp hashtable.cpp main.cpp linkedlist.h` and `hashtable.h` using the submit program. The directive is as follows;

```
submit -c csci212 -a 1 linkedlist.cpp hashtable.cpp main.cpp
                       linkedlist.h   hashtable.h
```

An extension of time for the completion of the assignment may be granted in certain circumstances. A request for an extension must be made to the Subject Coordinator before the due date. Late assignments without granted extension will be marked but the mark awarded will be reduced by 1 mark for each day late. Assignments will not be accepted more than three days late.