VNACCOUNT

User's Guide and Reference Manual

H. Ruotsalainen

September 1, 2000

Department of Physics and Astronomy
University of British Columbia
Supervisor: Dr. Matthew Choptuik
Student number: 200025466
Phone number: (604) 893-8976
E-mail: hkr@sfu.ca
Co-op course number: PHYS 436

VNACCOUNT: User's Guide and Reference Manual

H. Ruotsalainen

September 1, 2000

1 Co-op Summary

I worked in the physics department of the University of British Columbia for a professor. My supervisor's field of study is numerical relativity, or in other words, using computers to make models of interesting phenomena in the field of general relativity.

This semester I wrote a computer program called vnAccount which summarized the usage of a computing cluster set up and maintained by my supervisor. I also worked on various homework projects from a graduate course on general relativity taught by my supervisor.

2 Introduction

The vn computing cluster is a collection of computers available to students for large computing projects. A user can start multiple versions of their code running on the cluster instead of tying up their desktop machine. This is useful for anyone doing any kind of computationally intensive modeling of physical systems.

My supervisor was interested in knowing the level of usage of the cluster. It was to answer this question that I wrote the program vnAccount.

This report is meant to explain the purpose of the program vnAccount. It also serves as a user manual and reference on the logic of the code.

3 The computing cluster

The vn computing cluster consists of 64 dual-processor computing nodes and three front-end nodes. Each computing node has two Pentium III 450 MHz processors and 512 MBytes of RAM available for computing. Access to the computing nodes is through secure shell, a secure method of connecting to another computer over a network. Secure shell replaces telnet and rlogin. Users are free to start processes running on any node which has one free processor. The cluster is not intended to be used for parallel processing but more for distributed processing, allowing users to run several different simulations at once.

Data on the usage of the cluster is collected frequently, approximately every 5-10 minutes, using a program called top. Top is run on each computing node and the results are summarized into one file called an Rtop file. This file consists of two entries for each node identifying the two processes using the largest percentage of CPU time on that node. The entries are sorted by CPU usage. Listed information includes user name, the process running, the length of time the process has been running and percent memory usage, among other statistics. These Rtop files are collected into groups of approx 2000-3000 files and archived.

4 Function of the cluster program

The vnAccount program is used to summarize the usage of the vn computing cluster. It is written in the perl programming language. Perl is a language especially suited to be used for manipulating text and text files.

The program vnAccount “reads” the Rtop files to get information about processes that were started. When a new archive of Rtop files is available, the new Rtop files have to be added to a directory which contains all the Rtop files in the period of interest. The program then reads the Rtop files and for each unique process, saves the name of the user that started the process, the program name, run, process id number, runtime of the process, node used, date started and end date, if any. (see section 4.1 for more detailed information on how vnAccount works)

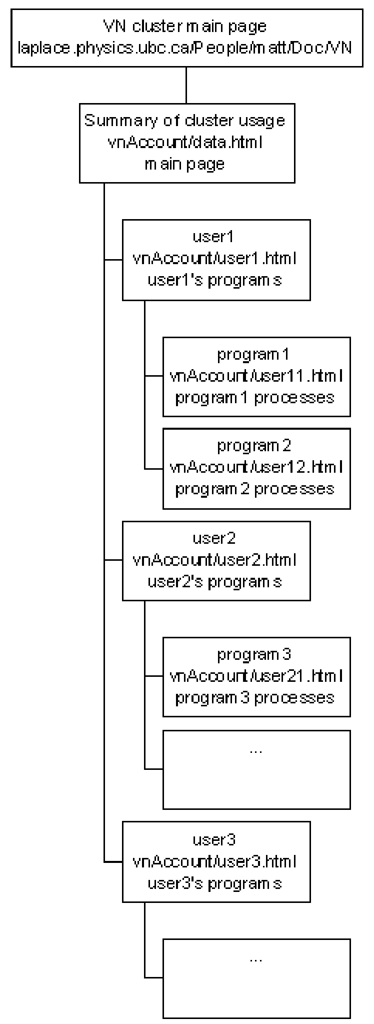
The processes are then sorted by user. The total amount of CPU time used by each user for all their processes and the number of processes started are calculated and then used to create a web page. The web page contains a graphical representation of the results and can be found at
http://laplace.physics.ubc.ca/People/matt/Doc/VN/vnAccount/data.html.

For each user the processes are further sorted by program name. The number of times each program was run and the total amount of time used for each program are calculated and this information is also posted on a web page, accessible from the first one by clicking on the user id. Finally, for any user, all the instances of a particular program are listed in according to runtime of the process. Included in this final list are the node used for each run, the process id number, the length of time the process has been running, the start date and the end date, if the process has ended before the last Rtop file was compiled. This information is accessible by clicking on the process name in the user's web page. Please see the site map on p. number. *include printouts of the top of the first page and one user page and one process page*

The main web page consists of a bar graph showing the usage of the cluster by user as well as the percentage of available CPU time used, the amount of available CPU time and the amount of time used for running processes. The length of the bars corresponds to the amount of time the user has used for his/her processes. This is reported as a percentage of the total time available for processing. The height of the bars corresponds to the number of processes started by the user. The taller the bar, the more processes the user has started. It is possible to see whether a user who is using a large amount of time has a large number of short-running processes or a small number of long-running processes.

Usage results The data on the vnAccount web page comes from Rtop files spanning Feb. 7, 2000 to Aug. 8, 2000. The typical usage for that time has been around 52-56%.

Site Map of web pages generated by vnAccount



5 Reference manual on vnAccount

5.1 How do I run vnAccount?

The code is run by typing vnAccount in the directory containing the Rtop files to be processed. There are three possible switches.

The switch "-help" prints out the help message and usage instruction (see below).

> This script calculates the usage statistics for linux cluster from the Rtop summary files. The results are summarized in HTML format. The main page is data.html
>
> The script must be called from the directory containing the Rtop files to be summarized
>
> usage is vnAccount [-p ¡percent¿] [-r ¡dirname¿] [-help]
>
> -p: sets the maxmimum value of the horizontal scale for the bar graphs. -p must be followed by a number between 0 and 100. If the -p switch is not used, the default maximum width of the bar graph is 15%.
>
> -r: stores the resulting html and temporary files in a directory. If a directory is specified then the script creates the directory "vnAccount" in the specified directory. All the specified directories must already exist. If a directory is not specified then the script attempts to create the directory "vnAccount" in the current directory. If that is not possible then the script attempts to create the directory "/tmp/logname/vnAccount" where "logname" is the user name of the current user. If that fails, the script exits. If the switch is not used the files are created in the current directory.
>
> -help: displays this text.
>
> Note: this script makes use of a GNU extension. It will run properly only on a Linux machine, not an SGI.

The switch "-r" without a directory name following it instructs the program to place any created files in a directory called vnAccount. It first attempts to make this directory in the current directory. Then, if that fails, it attempts to make the directory in the directory /tmp/logname, where "logname" is the username of the person running vnAccount. If that fails, it prints an error message and stops the program. If a directory name follows the switch, the program creates the directory vnAccount in the specified directory.

The switch "-p" sets the maximum percentage that is displayed in the graphs of the main web page. This switch should be followed by a number between 0 and 100. If it isn't, the program prints an error message and stops.

If any other switch is used, the program stops and prints usage instructions.

5.2 How does the code work?

Code algorithm

1. Read the Rtop files
    - 1.1 Sort by time created so that the oldest one is first
    - 1.2 Gothrough each file line by line
        - 1.2.1 Identify which lines are process lines and which ones are header lines
2. Store the information from those files
    - 2.1 Save all the information from any new user
        - 2.1.1 Create an entry in array "total_time"
            - 2.1.1.1 Save the time and userid
    - 2.1.2 Create a new array "userid"
            - 2.1.2.1 Save all process information
    - 2.1 Save all the information from any new process
        - 2.1.1 Update the usage time in "total_time"
        - 2.1.2 Create an entry in "userid"
    - 2.2 Update the time from any process that has already been identified
        - 2.2.1 Update usage in "total_time"
        - 2.2.2 Update time in "userid"
3. Sort and Summarize the information
    - 3.1 Normalize the time so that t=0 at the oldest Rtop file
        - 3.1.1 Identify which processes were in the oldest Rtop file
        - 3.1.2 Subtract the time recorded in the oldest Rtop file from the total time recorded for the process
    - 3.2 Calculate the time the cluster has been running
        - 3.2.1 Subtract the date of the newest Rtop file from the date of the oldest Rtop file
        - 3.2.2 Multiply by 2x64 (number of processors)
    - 3.3 Sort the usage information by runtime

3.3.1 Print the data to a file
3.3.2 Sort the file

3.4 Calculate the usage information as a percentage of total time available

3.5 Calculate how many processes each user started

3.6 Sort the user information by runtime

3.6.1 Print the data to a file
3.6.2 Sort the file

3.7 Sort the process information by runtime

3.7.1 Print the data to a file
3.7.2 Sort the file

3.8 Add all the time together for the total usage information

4. Write the web pages

4.1 Write the html for the main page

4.1.1 Write all the headers for data.html

4.1.1.1 Write the titles and total cluster usage information
4.1.1.2 Set up the tables

4.1.2 Write the usage information to data.html

4.1.2.1 Create a new table in each user's row
4.1.2.2 Use the percentage usage for length and the number of processes started for height
4.1.2.3 Make a link to user's page through the userid

4.1.3 Close data.html

4.2 Write the html for the user pages

4.2.1 Write all the headers for userid.html

4.2.1.1 Write the titles and usage information
4.2.1.2 Set up the tables

4.2.2 Write the user information to userid.html

4.2.2.1 Make a link to the program page through the process name

4.2.3 Close userid.html

4.3 Write the html for the process pages

4.3.1 Write the headers for userid#.html
4.3.2 Write the process information to userid#.html
4.3.3 Close userid#.html

Bugs, extra notes and future features Note that the time recorded in the Rtop files is CPU time, not actual time and it is half of what it should be. Because all the computing nodes are dual processor machines, the program top assumes both processors are working on a job and calculates the time assuming two processors are occupied. In fact, only one CPU works on any one job at a time and so it takes twice as much CPU time. The reported time is accordingly doubled when it is stored.

When the Rtop files were read, any process that was run by “root” or “idle” was ignored since they weren’t relevant to the usage of the cluster. Also if the process run was “top”, the information was ignored.

It was found that any one user rarely uses more than 15% of the available CPU time in the cluster so it was decided that the bar graph would show from 0-15% instead of 0-100%. This is adjustable using the -p switch.

Occasionally, negative runtimes show up in certain user’s process files. At this time, the reason for this is unknown.

The precision of the total CPU time available is a little excessive.

In a future version it would be nice to be able to compare CPU time used with “real” time used. This would give a measure of how efficient users’ programs are.

5.3 Code Listing

Please see http://laplace.physics.ubc.ca/People/hkr/vnAccount/vnAccount for the code listing.

```
#!/usr/bin/perl
#display helpfile
if ($ARGV[0] eq ‘‘-help’’) {
    print ‘‘\nThis script calculates the usage statistics for the linux cluster from
the Rtop\n’’;
    print "summary files. The results are summarized in HTML format. The main page
is\n";
    print "data.html\n\n";
    print "The script must be called from the directory containing the Rtop files to
\n";
    print "be summarized\n\n";
    print "usage is vnAccount [-p <percent>] [-r <dirname>] [-help]\n\n";
    print "-p: sets the maxmimum value of the horizontal scale for the bar graphs.\n";
    print "-p must be followed by a number between 0 and 100. If the -p switch is not
\n";
    print "used, the default maximum width of the bar graph is 15%.\n\n";
    print "-r: stores the resulting html and temporary files in a directory. If a\n";
    print "directory is specified then the script creates the directory \"vnAccount\"
in\n";
```

```
    print "the specified directory. All the specified directories must already exist.
\n";
    print "If a directory is not specified then the script attempts to create the
directory\n";
    print "\"vnAccount\" in the current directory. If that is not possible then the
script\n";
    print "attempts to create the directory \"/tmp/logname/vnAccount\" where
\"logname\" is\n";
    print "the user name of the current user. If that fails, the script exits. If the
\n";
    print "switch is not used the files are created in the current directory.\n\n";
    print "-help: displays this text.\n\n";
    print "Note: this script makes use of a GNU extension. It will run properly only
on a\n";
    print "Linux machine, not an SGI.\n\n";
}
#check for valid switches
elsif ($ARGV[0] && $ARGV[0] ne "-r" && $ARGV[0] ne "-p") {
    print "Incorrect switch. Type 'vnAccount -help' to see the switch options\n";
}
else {
    #set defaults
    $normalize = 15;
    $webdirect = ".";
    while (@ARGV) {
        $switch = shift @ARGV;
        next unless ($switch =~ /^-/);
        #check for the directory switch
        if ($switch eq "-r") {
            #if a directory is given, use that one
            if ($ARGV[0] && $ARGV[0] =~ /^[^-]/) {
                mkdir("$ARGV[0]/vnAccount", 0777);
                $webdirect = "$ARGV[0]/vnAccount";
            }
            else {
                #try to create the directory where vnAccount is called
                $webdirect = "vnAccount";
mkdir(vnAccount, 0777);
unless (open(TMEP, "+>vnAccount/tmep")) {
    #if that fails read the user name of the current user
    $logname = $ENV{LOGNAME};
    $webdirect = "/tmp/$logname/vnAccount";
    #try to create the directories /tmp/$logname and
/tmp/$logname/vnAccount
    mkdir ("/tmp/$logname", 0777);
```

```
    mkdir("/tmp/$logname/vnAccount", 0777);
}
#try to write to the created directory. If that fails, die.
open (TMEP, "+>$webdirect/tmep") || die "Can't create directory
$webdirect : $!\n";
    }
}
#set maximum value for horizontal scale on bar graph
elsif ($switch eq "-p") {
    #check to make sure "-p" is followed by a number between 0 and 100
    if ($ARGV[0] =~ /\d+/ && $ARGV[0] >= 0 && $ARGV[0] <= 100) {
$normalize = $ARGV[0];
    }
    else {
die "The switch -p must be followed by a number from 0-100\n";
    }
}
    }
    open (TMEP, "+>$webdirect/tmep") || die "Can't create files in directory $webdirect : $!\n";
#sort by time created, oldest first and list in a column
    open(DIRECT, "ls -1 | sort |");
    while(<DIRECT>) {
chop;
#make sure to pick up only the Rtop files (they start with the number 2)
if ($_ =~ /^[0-9]{4,4}/) {
    @lines[$i] = $_;
    $i = $i + 1;
}
    }

    $k = 0;
#time cluster has been running
    $cluster_time = 0;

#go through all the Rtop files, oldest first

    open (FIRST, $lines[0]);
    while(<FIRST>) {
#go through all the process lines
if ($_ =~ /^\d+/) {
    ($initial, $node, $pid, $user, $pri, $ni, $size, $rss, $share, $stat, $lib, $cpu, $mem, $time, $process) = split(/\s+/, $_);
    #ignore all "idle" or "root" processes
    unless ($user eq "idle" || $user eq "root" || $node =~ /^[^v]/ || $process eq "top") {
#distinguish between minutes and minutes:seconds form and turn both into total minutes
if ($time =~ /m/) {
```

```
    ($minutes) = split(/m/, $time);
    $minutes = 2 * $minutes;         #double time since top assumes both processors are working
}
else {
    ($minutes, $seconds) = split(/:/, $time);
    $minutes = 2*($minutes + ($seconds / 60));
}
#save the run-time of the first processes so that it can be subtracted off the total later
$initial_process{$pid} = $minutes . ":" . $user . ":" . $node;
$initial_time{$user} = ($initial_time{$user} + $minutes);

#create an array which holds the total time used by each user ($total_time)
#create an array which holds all the individual process data per user
if ($total_time{$user}) {     #update the arrays
    ($usage, $numprocess) = split(/:/, $total_time{$user});
    $$user{$pid} = $node . "+" . $process . "+" . $minutes . "+" . $date_time . "+" . $date_time;
    $usage = $usage + $minutes;
    $numprocess++;     #count the number of processes each user has started
    $total_time{$user} = $usage . ":" . $numprocess;
}
else {
    #create the relevant entries for each array if they don't already exist
    $$user{$pid} = $node . "+" . $process . "+" . $minutes . "+" . $date_time . "+" . $date_time;
    $total_time{$user} = $minutes . ":" . 1;
}
    }
}
#get date stamp on file
elsif ($_ =~ /^[a-z]/) {
    chop;
    ($address, $usage, $date_time) = split(/\s+/, $_, 3);
    $first_date = $date_time;
}
    }

    print $k, "\n";

#now process the rest of the files
    foreach $j (1..$#lines) {
$now = $lines[$j];
#open the current file
open(NOW, $now) || die "Can't open $now: $!\n";
while(<NOW>) {
    #split into pieces
#get all process info
```

```
    if ($_ =~ /^\d+/) {
($initial, $node, $pid, $user, $pri, $ni, $size, $rss, $share, $stat, $lib, $cpu, $mem, $time, $process) = split(/\s+/, $_);
#ignore all "idle" or "root" processes
unless ($user eq "idle" || $user eq "root" || $node =~ /^[^v]/ || $process eq "top") {
    #distinguish between minutes and minutes:seconds form and turn both into total minutes
    if ($time =~ /m/) {
($minutes) = split(/m/, $time);
$minutes = 2 * $minutes;         #double time since top assumes both processors are working
    }
    else {
($minutes, $seconds) = split(/:/, $time);
$minutes = 2*($minutes + ($seconds / 60));
    }
    #update the total_time and user arrays
    if ($total_time{$user}) {
($usage, $numprocess) = split(/:/, $total_time{$user});
if ($$user{$pid}) {
    ($master_node, $master_process, $master_time, $master_date, $stop_date) = split(/\+/, $$user{$pid}, 5);
    $$user{$pid} = $master_node. "+" . $master_process . "+" . $minutes . "+" . $master_date . "+" . $date_time;
    $usage = $usage + ($minutes - $master_time);
    $total_time{$user} = $usage . ":" . $numprocess;
}
else {
    $$user{$pid} = $node . "+" . $process . "+" . $minutes . "+" . $date_time . "+" . $date_time;
    $usage = $usage + $minutes;
    $numprocess++;
    $total_time{$user} = $usage . ":" . $numprocess;
}
    }
    else {
$$user{$pid} = $node . "+" . $process . "+" . $minutes . "+" . $date_time . "+" . $date_time;
$total_time{$user} = $minutes . ":" . 1;
    }
}
    }
#get date stamp on file
    elsif ($_ =~ /^[a-z]/) {
chop;
($address, $usage, $date_time) = split(/\s+/, $_, 3);
    }
}
#print the number of the file that was just read
print $j, " ";
    }
    print "\n";
```

```
#save the date of the oldest file
    $last_date = $date_time;
#print the results to a file
    open(TEMP, "+>$webdirect/temp");
    open(TMEP, "+>$webdirect/tmep");

    $all_user = 0;
#subtract the initial run time of processes in the first Rtop files
    foreach $user (sort keys(%total_time)) {
($usage, $numprocess) = split(/:/, $total_time{$user});
if ($initial_time{$user}) {
    $usage = ($usage - $initial_time{$user});
    foreach $pid (sort keys(%$user)) {
next unless ($initial_process{$pid});
($initial_minutes, $initial_user, $initial_node) = split(/:/, $initial_process{$pid});
($node, $process, $minutes, $date, $stop_date) = split(/\+/, $$user{$pid}, 5);
next unless ($user eq $initial_user && $node eq $initial_node);
$minutes = ($minutes - $initial_process{$pid});
$$user{$pid} = $node . "+" . $process . "+" . $minutes . "+" . $date . "+" . $stop_date;
    }
}
$all_user = ($all_user + $usage);
printf TEMP "%-7f  %-8s  %-5s\n", $usage, $numprocess, $user;
    }

    close(TEMP);

#sort by time, longest running at the top
    `sort -n -r -o$webdirect/temp $webdirect/temp`;

    open(TEMP, "$webdirect/temp");
    open(WEB, "+>$webdirect/data.html");

#take the first_date and last_date and calculate the number of minutes between them
#use the date system call to convert each date to the number of seconds since 00:00:00 Jan 1, 1970
#note: this is a GNU extension to date, it won't work on an SGI
    $firstdate = `date -d "$first_date" +%s`;
    $lastdate = `date -d "$last_date" +%s`;
    $cluster_time = (($lastdate-$firstdate)/60);
#calculate the total cluster time: the time the cluster has been running multiplied by the number of processors
    $cluster_time = 128*$cluster_time;
#hourmin converts from a time format of minutes to hrs:min format
#the first argument is time in minutes
    &hourmin(*cluster_time, *cluster_hours, *cluster_minutes);
    &hourmin(*all_user, *all_hours, *all_minutes);
```

```
    $total_perc = 100*($all_user/$cluster_time);

#start writing HTML code
    printf WEB "<HTML>\n<HEAD>\n<TITLE>Total time used for cluster processes</TITLE>\n</HEAD>\n\n<BODY BGCOLOR = \"white\">\n";
    printf WEB "<H1>Summary of <A HREF = \"http://laplace.physics.ubc.ca/People/matt/Doc/VN/index.html\">Cluster</A> Usage</H1>\n";
    printf WEB "<I>Note: The timings and statistics rooted in this document are currently for informational purposes only, and are not guaranteed to be completely correct. They do provide, however, a representative view of overall cluster usage.</I><BR><BR>\n";
    printf WEB "<B>Percentage of total cluster time used = %1.1f\%</B><BR>\nTotal cluster time available (hrs : min): %1s:%02d<BR>\nTotal time used for running processes (hrs : min) = %1s:%02d<BR>\n<BR>\n",$total_perc, $cluster_hours, $cluster_minutes, $all_hours, $all_minutes;
    printf WEB "The time used by each user is displayed as a percentage of the total cluster time available. The width of the bar corresponds to the percentage of
time used.\n";
    printf WEB "The height of the bar corresponds to the number of processes run by that user.\n";
    printf WEB "Click on the user name on the right to see each user’s process information.<BR>\n<BR>\n";
    printf WEB "Processes counted between <B>%1s</B> and <B>%1s</B>\n", $first_date, $last_date;
#create the main table in the main page
    printf WEB "<TABLE BORDER = \"1\" WIDTH = \"100\%\">\n\t<TR>\n\t<TD><TABLE WIDTH = \"100\%\">\n\t\t<TR><TD ALIGN=\"LEFT\">0</TD>\n\t\t<TD ALIGN=\"RIGHT\">%1s\% </TD></TR>\n\t</TABLE></TD>\n\t<TD>Percentage of time used / User / No. processes</TD>\n\t</TR>\n", $normalize;

#TEMP is the file that contains all the total_time data sorted by each user’s usage time
    while(<TEMP>) {
chop;
($user_time,$numprocess,$user) = split(/\s+/, $_);
$report_percent = 100*($user_time/($cluster_time));     #do the calculations for the lengths of the bar graphs
$display_percent = 100*($report_percent/$normalize);
#convert time back to hours:minutes format
&hourmin(*user_time, *user_hours, *user_minutes);
#do the calculations for the height of the bar graphs. Height goes as log10 of the number of processes except if number of processes = 1 then height = 1 as well
if ($numprocess <= 1) {
    $height = 1;
}
else {
    $height = 30*(log($numprocess)/2.302585093);
}
#create a table in each row of the main table. The length of the inner table is set by percent: this creates the bar graph appearance
#both the heights of the main table row and the inner table have to be set the same to get the height difference
printf WEB "\t<TR HEIGHT = %1s>\n\t<TD WIDTH = \"70\%\"><TABLE HEIGHT = %1s BORDER = \"0\" BGCOLOR = \"black\" WIDTH = \"%1d\%\">\n\t\t<TR><TD><SMALL> </SMALL></TD></TR>\n\t</TABLE></TD>\n\t<TD><A HREF=\"%1s.html\">%1.1f\% / %1s / %1s</A></TD>\n\t</TR>\n", $height, $height, $display_percent, $user, $report_percent, $user, $numprocess;
open (TEMP1, "+>$webdirect/temp1");
#print to TEMP1 for sorting by process for each user
foreach $pid (sort keys(%$user)) {
    ($master_node, $master_process, $master_time, $master_date, $stop_date) = split(/\+/, $$user{$pid}, 5);
    printf TEMP1 "%-12s  %-7f  %-5s  %-4s  %-28s+%28s\n", $master_process, $master_time, $pid, $master_node, $master_date, $stop_date;
}

close (TEMP1);
‘sort -o$webdirect/temp1 $webdirect/temp1‘;
open (TEMP1, "$webdirect/temp1");
open (TEMP2, "+>$webdirect/temp2");
open (PERSON, "+>$webdirect/$user.html");
```

```
printf PERSON "<HTML>\n<HEAD>\n<TITLE>%ls's process information</TITLE>\n</HEAD>\n\n<BODY>\n", $user;
printf PERSON "Click on the process to see the usage details for that process\n<BR>\n%1s used %1s:%02d (hrs : min) of processing time\n<BR><BR>\n", $user, $user_hours, $user_minutes;
printf PERSON "<TABLE>\n\t<TR ALIGN = \"LEFT\">\n\t<TH>USER  </TH>\n\t<TH>PROCESS  </TH>\n\t<TH>NUMBER OF PROCESSES  </TH>\n\t<TH>TIME USED FOR PROCESS  </TH>\n\t</TR>\n";
#initialize lastprocess to something unlikely to be an actual process name
#numproc counts the number of different processes run by each user; used to create a unique filename for PROCESS
$lastprocess = 1;
$numproc = 0;
#TEMP1 is the file that contains all the process data for one user sorted by amount of time used for each process
open (FINAL, "+>$webdirect/final");
while (<TEMP1>) {
    chop;
    ($temp_process, $temp_time, $temp_pid, $temp_node, $temp_date) = split(/\s+/, $_, 5);
    $nextprocess = $temp_process;
    #if this is the first line in TEMP1 (no prior process) or if the process hasn't changed then print the info to TEMP2 and reassign the lastprocess
    if ($nextprocess eq $lastprocess || $lastprocess eq 1) {
printf TEMP2 "%-7f  %5s  %-12s  %-4s  %-56s\n", $temp_time, $temp_pid, $temp_process, $temp_node, $temp_date;
$lastprocess = $nextprocess;
    }
    #when the process changes: close TEMP2; sort it by time, longest running process first
    else {
#procnum counts the number of times each process was run; printed in PERSON
$procnum = 0;
$process_time = 0;
$numproc++;
close (TEMP2);
`sort -n -r -o$webdirect/temp2 $webdirect/temp2`;
open (TEMP2, "$webdirect/temp2");
open (PROCESS, "+>$webdirect/$user$numproc");
printf PROCESS "USER: %-8s\n\n", $user;
printf PROCESS "       PROCESS       TIME (hrs:min)  PID    NODE  DATE STARTED                  DATE STOPPED\n";
#TEMP2 is the file that contains the process data for one process and one user sorted by time
#open the text file for that process; read the lines from TEMP2 and print to PROCESS
while (<TEMP2>) {
    chop;
    ($master_time, $master_pid, $master_process, $master_node, $master_date) = split(/\s+/, $_, 5);
    ($start_date, $stop_date) = split(/\+/, $master_date, 2);
    $process_time = $process_time + $master_time;
    &hourmin(*master_time, *master_hours, *master_minutes);
    $procnum++;
    if ($stop_date ne $last_date) {
printf PROCESS "%4s:  %-12s  %11s:%02d  %-5s  %-4s  %-28s  %-28s\n", $procnum, $master_process, $master_hours, $master_minutes, $master_pid, $master_node, $start_date, $stop_date;
    }
    else {
printf PROCESS "%4s:  %-12s  %11s:%02d  %-5s  %-4s  %-28s         -\n", $procnum, $master_process, $master_hours, $master_minutes, $master_pid, $master_node, $start_date;
    }
```

```
}
#update the HTML file for the user which lists the processes
printf FINAL "%7f  %-8s  %-12s  %2s  %2s\n", $process_time, $user, $master_process, $procnum, $numproc;
close (TEMP2);
close (PROCESS);
#open a new TEMP2 and print the new information; reassign lastprocess
open (TEMP2, "+>$webdirect/temp2");
printf TEMP2 "%-7f  %5s  %-12s  %-4s  %-28s\n", $temp_time, $temp_pid, $temp_process, $temp_node, $temp_date;
$lastprocess = $nextprocess;
    }
}
#the last process in TEMP1 won't go through the else section above so now do: sort TEMP2 by time, print PROCESS and update PERSON
$procnum = 0;
$process_time = 0;
$numproc++;

close (TEMP2);
`sort -n -r -o$webdirect/temp2 $webdirect/temp2`;
open (TEMP2, "$webdirect/temp2");
open (PROCESS, "+>$webdirect/$user$numproc");
printf PROCESS "USER: %-8s\n\n", $user;
printf PROCESS "       PROCESS       TIME (hrs:min)  PID    NODE  DATE STARTED                  DATE STOPPED\n";
while (<TEMP2>) {
    chop;
    ($master_time, $master_pid, $master_process, $master_node, $master_date) = split(/\s+/, $_, 5);
    ($start_date, $stop_date) = split(/\+/, $master_date, 2);
    $process_time = $process_time + $master_time;
    &hourmin(*master_time,*master_hours,*master_minutes);
    $procnum++;
    if ($stop_date ne $last_date) {
printf PROCESS "%4s:  %-12s  %11s:%02d  %-5s  %-4s  %-28s  %-28s\n", $procnum, $master_process, $master_hours, $master_minutes, $master_pid, $master_node, $start_date, $stop_date;
    }
    else {
printf PROCESS "%4s:  %-12s  %11s:%02d  %-5s  %-4s  %-28s          -\n", $procnum, $master_process, $master_hours, $master_minutes, $master_pid, $master_node, $start_date;
    }
}
printf FINAL "%7f  %-8s  %-12s  %2s  %2s\n", $process_time, $user, $master_process, $procnum, $numproc;
close (FINAL);
`sort -n -r -o$webdirect/final $webdirect/final`;
close (TEMP2);
close (PROCESS);
open (FINAL, "$webdirect/final");
while (<FINAL>) {
    chop;
    ($process_time, $user, $master_process, $procnum, $numproc) = split (/\s+/, $_);
```

```
    &hourmin(*process_time, *process_hours, *process_minutes);
    printf PERSON "\t<TR ALIGN = \"LEFT\">\n\t<TD>%1s  </TD>\n\t<TD><A HREF = \"%1s%1s\">%1s</A>  </TD>\n\t<TD>%1s  </TD>\n\t<TD>%1s:%02d  </TD>\n\t</TR>\n",$user,$user,$numproc,$master_process, $procnum, $process_hours, $process_minutes;
}
#finish off the HTML for PERSON
printf PERSON "</TABLE>\n<BR>\n";
printf PERSON "<A HREF=\"data.html\">Back to total time graph</A>\n</BODY>\n</HTML>\n";
close (PERSON);
    }

#finish off the HTML for WEB
    require "ctime.pl";
    $today = &ctime(time);
    printf WEB "</TABLE><BR>\n<BR>\n<HR>Generated on <I>%1s</I> by <B><A HREF=\"http://laplace.physics.ubc.ca/People/hkr/vnAccount/vnAccount\">vnAccount</A></B>.<BR>\n", $today;
    printf WEB "Copyright 2000 by <A HREF=\"http://laplace.physics.ubc.ca/People/hkr\">Hanna Ruotsalainen</A>.<BR>\n";
    printf WEB "Supported by <A HREF=\"http://www.ciar.ca\">CIAR</A> and <A HREF = \"http://www.nserc.ca\">NSERC</A><BR>\n";
    printf WEB"</BODY>\n</HTML>";
    print "The files were put in directory ", $webdirect, "\n";
    print "The main page is called data.html\n";
}


#subroutine which changes from minutes format to hrs:min format
sub hourmin {
    #minute_time is passed to the subroutine, hours and minutes are altered in the routine and then accessed back at the subroutine call
    local(*minute_time, *hours, *minutes) = @_;
    local ($seconds);
    $hours_float = ($minute_time/60);
    #split on the decimal point
    ($minutes, $seconds) = split(/\./, $minute_time);
    ($hours) = split(/\./, $hours_float);
    #round the seconds up or down to the nearest minute
    if ($seconds > 50) {
#use mod for the minutes; returns the remainder from minutes/60
$minutes = ($minutes%60)+1;
if ($minutes > 59) {
    $hours++;
    $minutes = 0;
}
    }
    else {
$minutes = ($minutes%60);
    }
}
```