

Process in unix

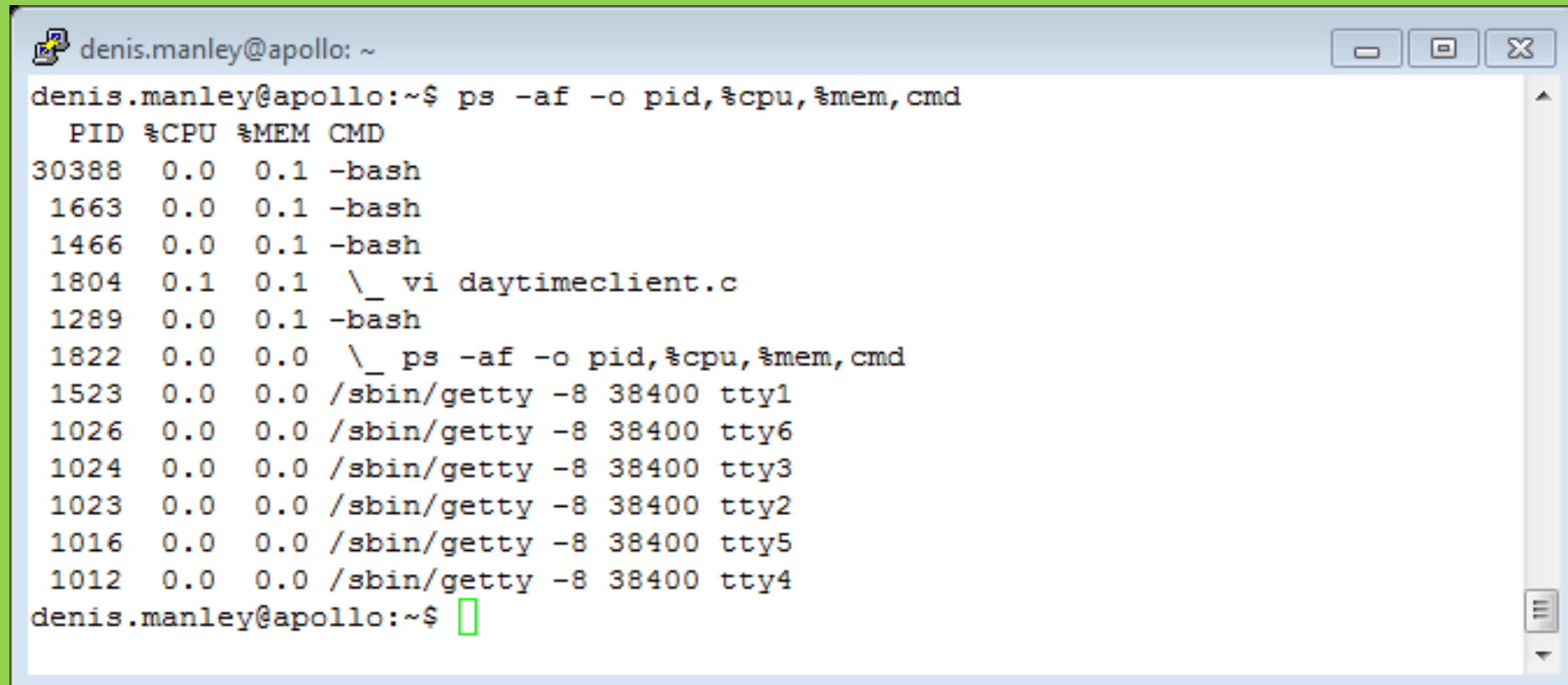
Create ; synchronize and exec
commands

Process in Unix

- The `ps` command shows basic information about the PCBs
- **PID time DESCRIPTION (CMD)**
- e.g. the `man ps` shows other command line arguments; e.g.

- | | | | |
|---|-------------------|-------------------|--|
| • | <code>%cpu</code> | <code>%CPU</code> | cpu utilization of the process in "##.#" format. |
| • | | | Currently, it is the CPU time used divided by the time the |
| • | | | process has been running (cputime/realtime ratio), |
| • | | | expressed as a percentage. It will not add up to 100% |
| • | | | unless you are lucky. (alias <code>pcpu</code>). |
| | | | |
| • | <code>%mem</code> | <code>%MEM</code> | ratio of the process's resident set size to the physical |
| • | | | memory on the machine, expressed as a percentage. |
| • | | | (alias <code>pmem</code>). |

Example more detailed ps command



A terminal window titled 'denis.manley@apollo: ~' showing the output of the command 'ps -af -o pid,%cpu,%mem,cmd'. The output lists several processes, including multiple instances of bash and getty, and one instance of vi editing daytimeclient.c. The terminal window has standard Linux window controls (minimize, maximize, close) in the top right corner and a scrollbar on the right side.

```
denis.manley@apollo:~$ ps -af -o pid,%cpu,%mem,cmd
  PID %CPU %MEM    CMD
 30388   0.0   0.1  -bash
  1663   0.0   0.1  -bash
  1466   0.0   0.1  -bash
  1804   0.1   0.1  \_ vi daytimeclient.c
  1289   0.0   0.1  -bash
  1822   0.0   0.0  \_ ps -af -o pid,%cpu,%mem,cmd
 1523   0.0   0.0  /sbin/getty -8 38400 tty1
 1026   0.0   0.0  /sbin/getty -8 38400 tty6
 1024   0.0   0.0  /sbin/getty -8 38400 tty3
 1023   0.0   0.0  /sbin/getty -8 38400 tty2
 1016   0.0   0.0  /sbin/getty -8 38400 tty5
 1012   0.0   0.0  /sbin/getty -8 38400 tty4
denis.manley@apollo:~$
```

Top commands in linux: list of processes

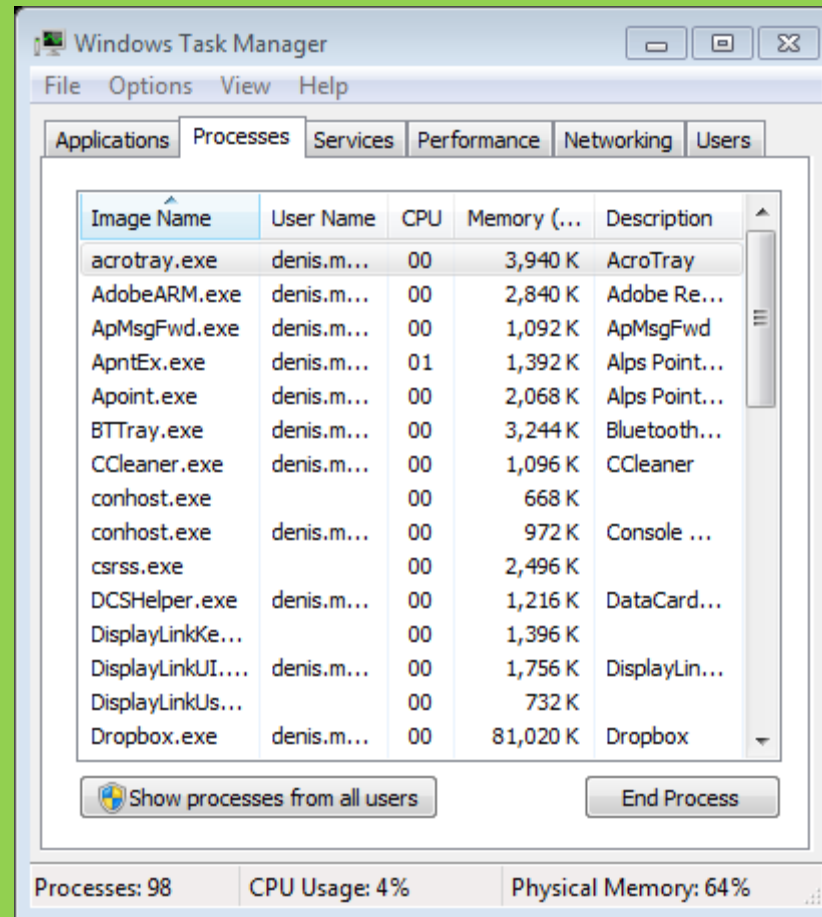
```
denis.manley@apollo: ~
top - 15:38:57 up 4 days, 6:26, 4 users, load average: 0.00, 0.01, 0.05
Tasks: 107 total, 1 running, 106 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.0 us, 0.0 sy, 0.0 ni, 100.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem: 4048192 total, 1281008 used, 2767184 free, 157532 buffers
KiB Swap: 1044476 total, 0 used, 1044476 free. 636160 cached Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1	root	20	0	43088	3652	2060	S	0.0	0.1	0:02.80	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.02	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:01.00	ksoftirqd/0
5	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/0:+
7	root	20	0	0	0	0	S	0.0	0.0	0:07.17	rcu_sched
8	root	20	0	0	0	0	S	0.0	0.0	0:11.11	rcuos/0
9	root	20	0	0	0	0	S	0.0	0.0	0:03.13	rcuos/1
10	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_bh
11	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcuob/0
12	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcuob/1
13	root	rt	0	0	0	0	S	0.0	0.0	0:00.26	migration/0
14	root	rt	0	0	0	0	S	0.0	0.0	0:02.06	watchdog/0
15	root	rt	0	0	0	0	S	0.0	0.0	0:01.90	watchdog/1
16	root	rt	0	0	0	0	S	0.0	0.0	0:00.30	migration/1
17	root	20	0	0	0	0	S	0.0	0.0	0:17.27	ksoftirqd/1
19	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/1:+
20	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	khelper
21	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kdevtmpfs
22	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	netns
23	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	writeback
24	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kintegrityd
25	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	bioset
26	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/u5+
27	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kblockd

htop command in linux

```
denis.manley@apollo: ~  
  
1  [|||] 2.0%] Tasks: 45, 106 thr; 1 running  
2  [ 0.0%] Load average: 0.00 0.01 0.05  
Mem[|||||] 477/3953MB] Uptime: 4 days, 06:28:26  
Swp[ 0/1019MB]  
  
PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command  
1835 denis.man 20 0 39640 3168 2216 R 0.7 0.1 0:00.66 htop  
1349 root 20 0 155M 4064 3096 S 0.0 0.1 3:03.42 /usr/sbin/vmtools  
1128 root 20 0 19188 788 520 S 0.0 0.0 0:59.57 /usr/sbin/irqbala  
1288 denis.man 20 0 321M 4736 1328 S 0.0 0.1 0:00.13 sshd: denis.manle  
1 root 20 0 43088 3652 2060 S 0.0 0.1 0:02.80 /sbin/init  
338 root 20 0 19472 656 456 S 0.0 0.0 0:00.12 upstart-udev-brid  
352 root 20 0 51252 1560 984 S 0.0 0.0 0:00.25 /lib/systemd/syst  
503 root 20 0 15256 408 200 S 0.0 0.0 0:00.04 upstart-socket-br  
807 root 20 0 319M 10528 7944 S 0.0 0.3 0:20.96 smbd -F  
865 root 20 0 15272 404 200 S 0.0 0.0 0:00.02 upstart-file-brid  
871 root 20 0 226M 2996 1352 S 0.0 0.1 0:18.94 nmbd -D  
914 messagebu 20 0 49228 1756 864 S 0.0 0.0 0:00.44 dbus-daemon --sys  
919 syslog 20 0 259M 2672 1752 S 0.0 0.1 0:00.09 rsyslogd  
920 syslog 20 0 259M 2672 1752 S 0.0 0.1 0:00.00 rsyslogd  
921 syslog 20 0 259M 2672 1752 S 0.0 0.1 0:00.14 rsyslogd  
916 syslog 20 0 259M 2672 1752 S 0.0 0.1 0:00.26 rsyslogd  
937 root 20 0 301M 2808 636 S 0.0 0.1 0:00.00 smbd -F  
958 root 20 0 53156 2904 2184 S 0.0 0.1 0:00.45 /lib/systemd/syst  
1012 root 20 0 15812 936 784 S 0.0 0.0 0:00.00 /sbin/getty -8 38  
1016 root 20 0 15812 940 784 S 0.0 0.0 0:00.00 /sbin/getty -8 38  
1018 root 20 0 319M 3456 872 S 0.0 0.1 0:02.22 smbd -F  
1023 root 20 0 15812 940 784 S 0.0 0.0 0:00.00 /sbin/getty -8 38  
1024 root 20 0 15812 936 784 S 0.0 0.0 0:00.00 /sbin/getty -8 38  
F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice -F8Nice +F9Kill F10Quit
```

Process details in windows



Job and Process States

- Status changes: as a job or process moves through the system
 - HOLD and placed in a queue (sometimes a priority queue)
 - READY (job begins processing if enough resources available)
 - WAITING (can not continue until specific resource available e.g. i/o request)
 - RUNNING: processing
 - FINISHED

Process Creation: Unix

- In Unix, processes are created using `fork()`
- Creates and initializes a PCB and creates a new address space
- Initializes the address space with a copy of the entire contents of the address space of the parent
- Initializes the kernel resources to point to the resources used by parent (e.g., open files)
- Places the child PCB on the ready queue
- Fork returns twice: Returns the child's PID to the parent, "0" to the child

Process Creation

- Address space
 - Child duplicate of parent
 - Child has a program (process) loaded into it
- UNIX/Linux examples
 - The `fork()` system call in Unix creates a new process. After a successful `fork()` call, *two copies* of the original code will be running. In the original process (the parent) the return value of `fork()` will be the process ID of the child. In the new child process the return value of `fork()` will be 0

Synchronization

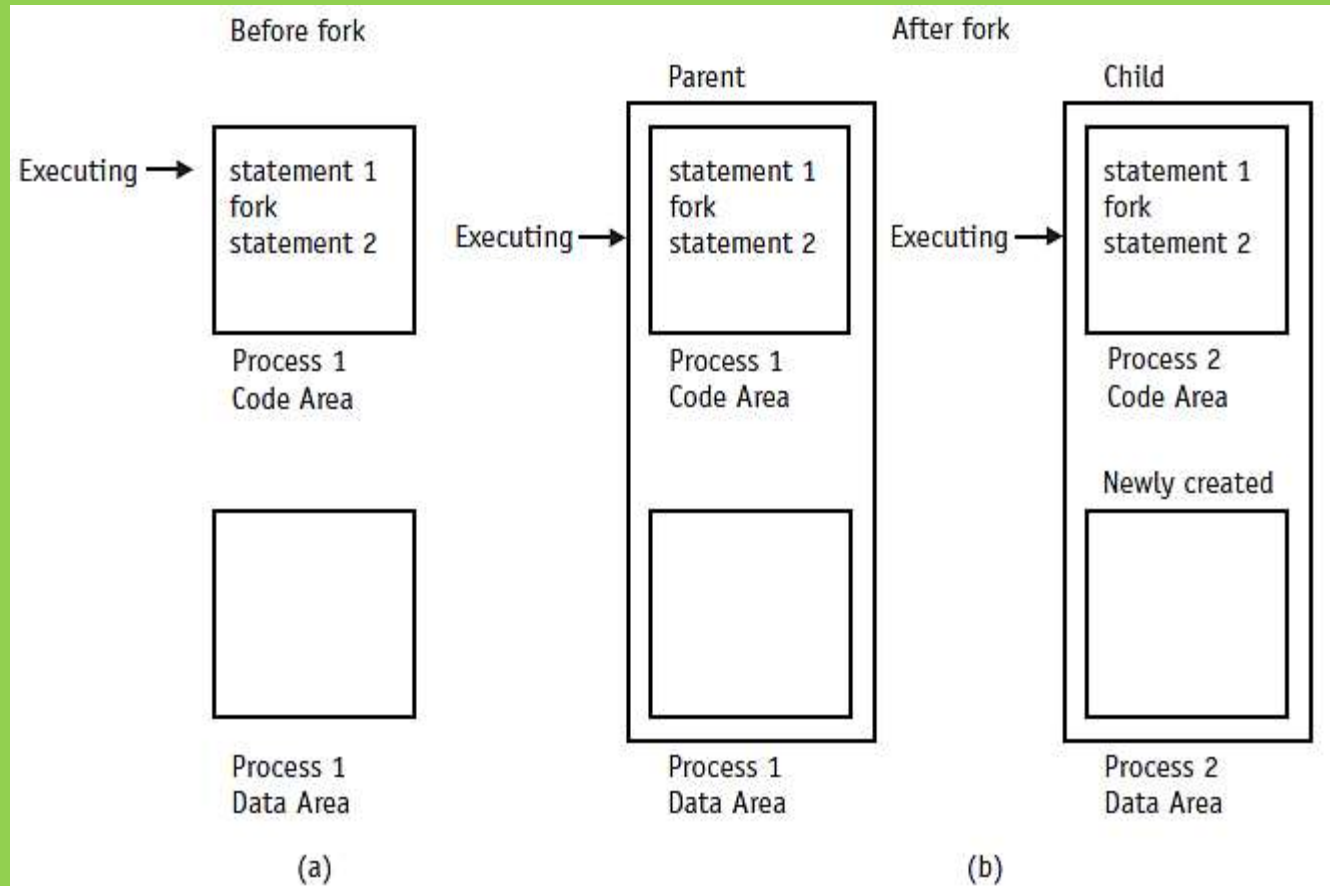
- **Fork()**
 - UNIX command: generates one program from another program
 - Gives second program all first program attributes
 - code, global data, heap and stack, registers (including *program counter*), and open files.
 - Saves first program in original form
 - Splits program: two versions of the parent (parent and child)
 - Both run from statement after `fork` command
 - `fork` executed
 - “Process id” (pid) generated
 - Ensures each process has unique ID number

Synchronization (cont'd.)

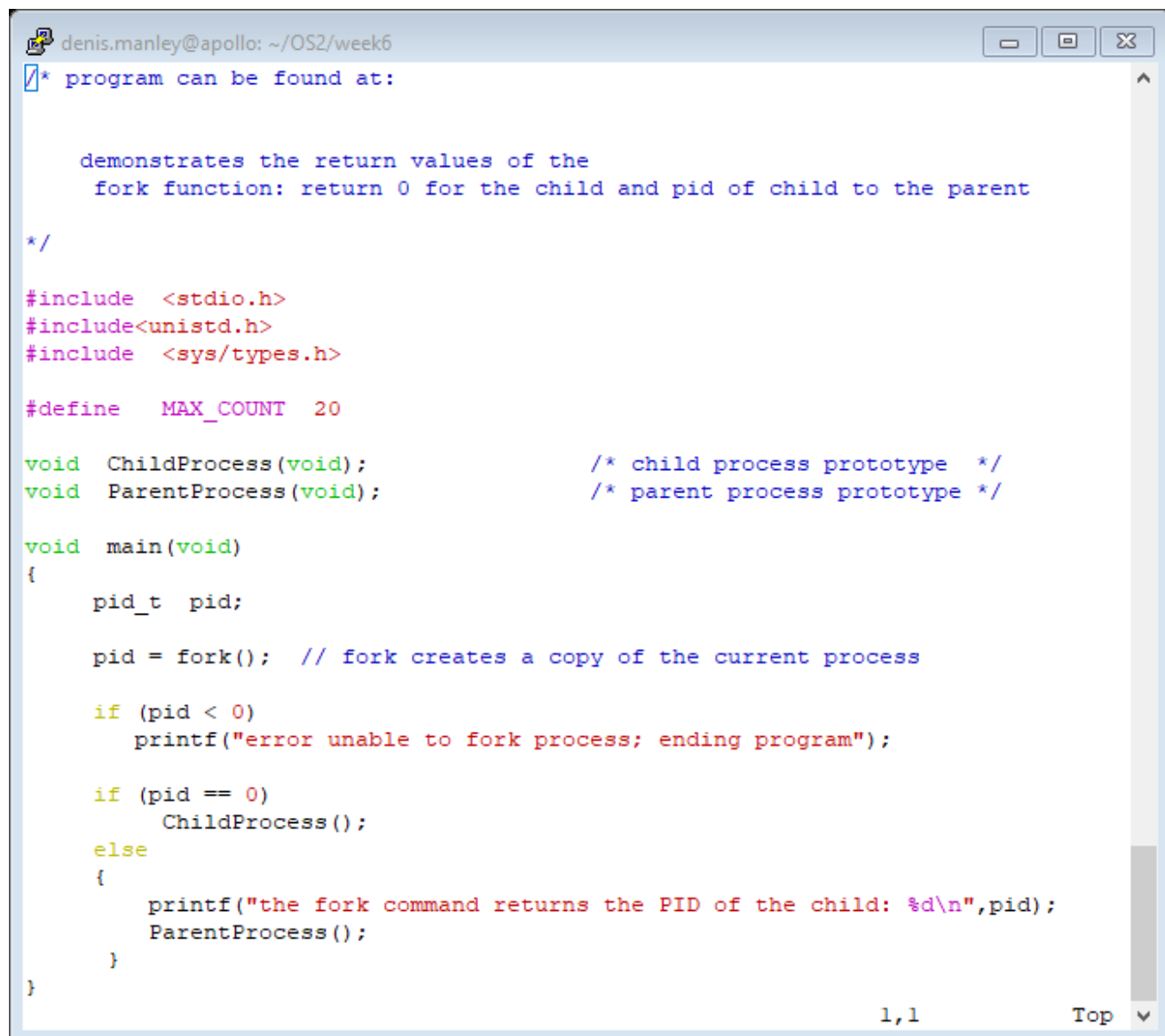
when the fork command is received, the parent process:

shown in (a) begets the child process shown

in (b) and Statement 2 (one of the child and one of the parent) is executed twice.



Fork1.c main



```
denis.manley@apollo: ~/OS2/week6
/* program can be found at:

    demonstrates the return values of the
    fork function: return 0 for the child and pid of child to the parent
*/

#include <stdio.h>
#include<unistd.h>
#include <sys/types.h>

#define MAX_COUNT 20

void ChildProcess(void);          /* child process prototype */
void ParentProcess(void);        /* parent process prototype */

void main(void)
{
    pid_t pid;

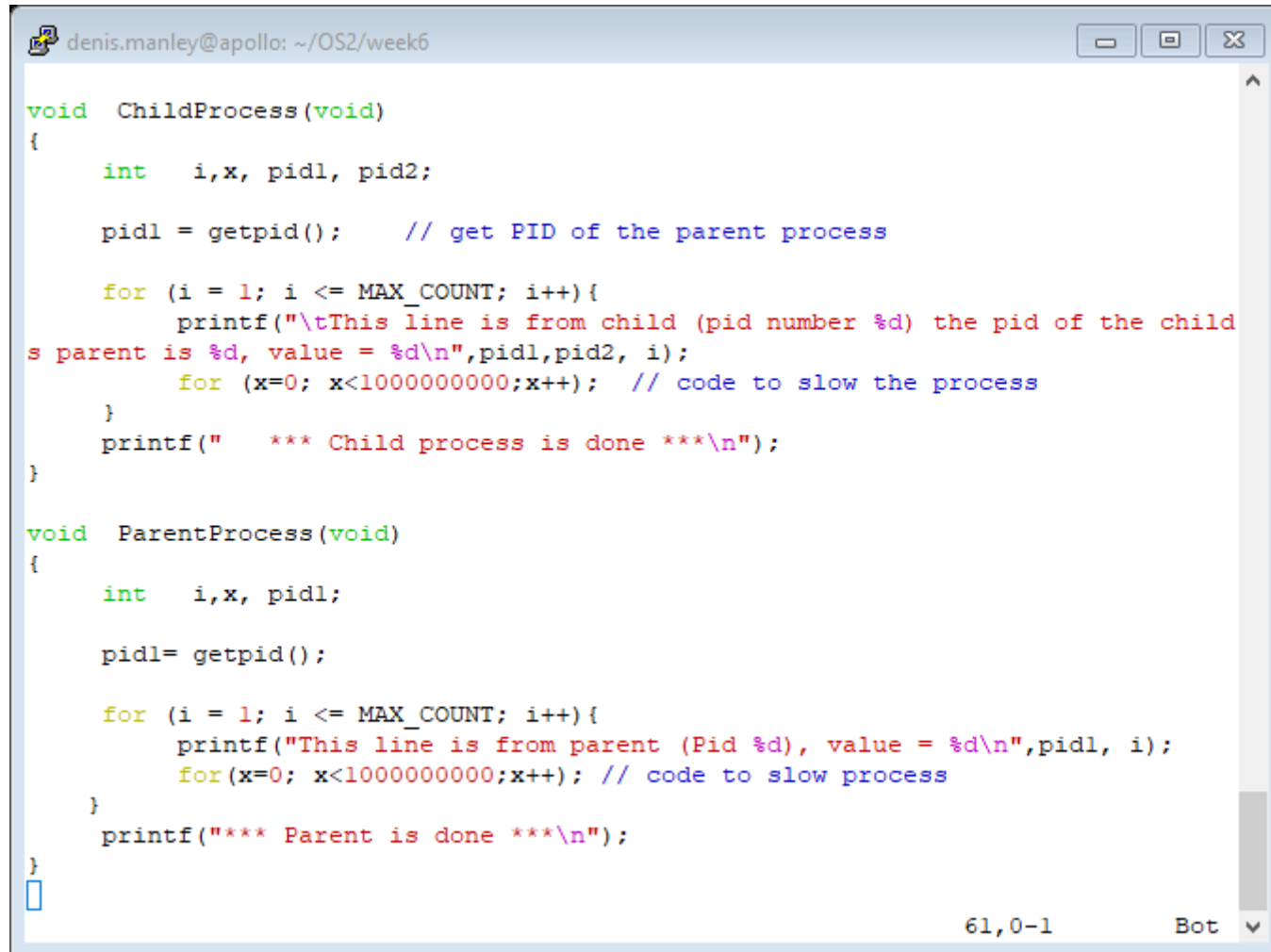
    pid = fork(); // fork creates a copy of the current process

    if (pid < 0)
        printf("error unable to fork process; ending program");

    if (pid == 0)
        ChildProcess();
    else
    {
        printf("the fork command returns the PID of the child: %d\n",pid);
        ParentProcess();
    }
}
```

1,1 Top

A simply program for fork: Fork1.c



```
denis.manley@apollo: ~/OS2/week6

void ChildProcess(void)
{
    int i,x, pid1, pid2;

    pid1 = getpid();    // get PID of the parent process

    for (i = 1; i <= MAX_COUNT; i++){
        printf("\tThis line is from child (pid number %d) the pid of the child
s parent is %d, value = %d\n",pid1,pid2, i);
        for (x=0; x<1000000000;x++); // code to slow the process
    }
    printf("    *** Child process is done ***\n");
}

void ParentProcess(void)
{
    int i,x, pid1;

    pid1= getpid();

    for (i = 1; i <= MAX_COUNT; i++){
        printf("This line is from parent (Pid %d), value = %d\n",pid1, i);
        for(x=0; x<1000000000;x++); // code to slow process
    }
    printf("*** Parent is done ***\n");
}
```

A sample Output Fork1.c

```
denis.manley@soc-apollo: ~/OS2/week6
This line is from child, value = 2
This line is from parent, value = 5
  This line is from child, value = 3
This line is from parent, value = 6
  This line is from child, value = 4
This line is from parent, value = 7
This line is from parent, value = 8
  This line is from child, value = 5
This line is from parent, value = 9
  This line is from child, value = 6
This line is from parent, value = 10
  This line is from child, value = 7
This line is from parent, value = 11
  This line is from child, value = 8
This line is from parent, value = 12
  This line is from child, value = 9
This line is from parent, value = 13
  This line is from child, value = 10
This line is from parent, value = 14
  This line is from child, value = 11
This line is from parent, value = 15
  This line is from child, value = 12
This line is from parent, value = 16
  This line is from child, value = 13
This line is from parent, value = 17
  This line is from child, value = 14
This line is from parent, value = 18
  This line is from child, value = 15
This line is from parent, value = 19
  This line is from child, value = 16
This line is from parent, value = 20
  This line is from child, value = 17
*** Parent is done ***
  This line is from child, value = 18
  This line is from child, value = 19
  This line is from child, value = 20
  *** Child process is done ***
denis.manley@soc-apollo:~/OS2/week6$
```

Modify Fork1.c

- Reduce the number of iterations in the for loop: Fork1.c and Explain the effect.
- Use ubuntu app/virtual machine and test the Fork1.c code: is it slower or faster?

Fork2.c main function

```
denis.manley@apollo: ~/OS2/week6

/*  fork2.c similiar to fork1.c but uses the sleep command to synchronise the parent and the child */
/

[]

#include <stdio.h>
#include<unistd.h>
#include <sys/types.h>

#define  MAX_COUNT  20

void  ChildProcess(void);          /* child process prototype */
void  ParentProcess(void);        /* parent process prototype */

void  main(void)
{
    pid_t  pid;

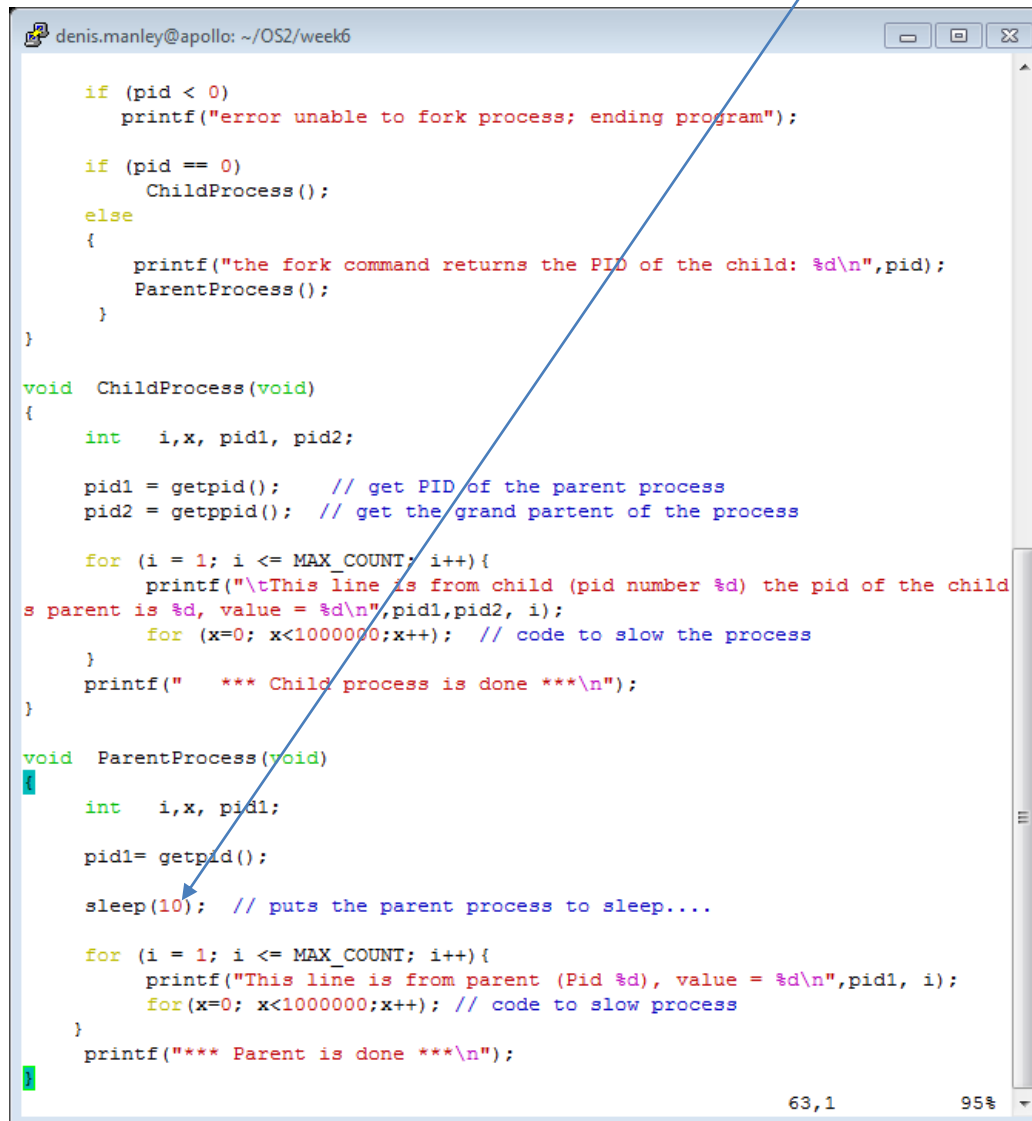
    pid = fork(); // fork creates a copy of the current process

    if (pid < 0)
        printf("error unable to fork process; ending program");

    if (pid == 0)
        ChildProcess();
    else
    {
        printf("the fork command returns the PID of the child: %d\n",pid);
        ParentProcess();
    }
}
```

4,0-1 Top

Fork2.c: Synchronise output with **sleep()** command:



```
denis.manley@apollo: ~/OS2/week6

if (pid < 0)
    printf("error unable to fork process; ending program");

if (pid == 0)
    ChildProcess();
else
{
    printf("the fork command returns the PID of the child: %d\n",pid);
    ParentProcess();
}

void ChildProcess(void)
{
    int i,x, pid1, pid2;

    pid1 = getpid(); // get PID of the parent process
    pid2 = getppid(); // get the grand parent of the process

    for (i = 1; i <= MAX_COUNT; i++){
        printf("\tThis line is from child (pid number %d) the pid of the child
parent is %d, value = %d\n",pid1,pid2, i);
        for (x=0; x<1000000;x++); // code to slow the process
    }
    printf("    *** Child process is done ***\n");
}

void ParentProcess(void)
{
    int i,x, pid1;

    pid1= getpid();

    sleep(10); // puts the parent process to sleep....

    for (i = 1; i <= MAX_COUNT; i++){
        printf("This line is from parent (Pid %d), value = %d\n",pid1, i);
        for(x=0; x<1000000;x++); // code to slow process
    }
    printf("*** Parent is done ***\n");
}
```

Fork2.c: Synchronise output with sleep command:

```
denis.manley@apollo: ~/OS2/week6
the fork command returns the PID of the child: 1616
  This line is from child (pid number 1616) the pid of the childs parent is 1615, value = 1
  This line is from child (pid number 1616) the pid of the childs parent is 1615, value = 2
  This line is from child (pid number 1616) the pid of the childs parent is 1615, value = 3
  This line is from child (pid number 1616) the pid of the childs parent is 1615, value = 4
  This line is from child (pid number 1616) the pid of the childs parent is 1615, value = 5
  This line is from child (pid number 1616) the pid of the childs parent is 1615, value = 6
  This line is from child (pid number 1616) the pid of the childs parent is 1615, value = 7
  This line is from child (pid number 1616) the pid of the childs parent is 1615, value = 8
  This line is from child (pid number 1616) the pid of the childs parent is 1615, value = 9
  This line is from child (pid number 1616) the pid of the childs parent is 1615, value = 10
  This line is from child (pid number 1616) the pid of the childs parent is 1615, value = 11
  This line is from child (pid number 1616) the pid of the childs parent is 1615, value = 12
  This line is from child (pid number 1616) the pid of the childs parent is 1615, value = 13
  This line is from child (pid number 1616) the pid of the childs parent is 1615, value = 14
  This line is from child (pid number 1616) the pid of the childs parent is 1615, value = 15
  This line is from child (pid number 1616) the pid of the childs parent is 1615, value = 16
  This line is from child (pid number 1616) the pid of the childs parent is 1615, value = 17
  This line is from child (pid number 1616) the pid of the childs parent is 1615, value = 18
  This line is from child (pid number 1616) the pid of the childs parent is 1615, value = 19
  This line is from child (pid number 1616) the pid of the childs parent is 1615, value = 20
*** Child process is done ***
This line is from parent (Pid 1615), value = 1
This line is from parent (Pid 1615), value = 2
This line is from parent (Pid 1615), value = 3
This line is from parent (Pid 1615), value = 4
This line is from parent (Pid 1615), value = 5
This line is from parent (Pid 1615), value = 6
This line is from parent (Pid 1615), value = 7
This line is from parent (Pid 1615), value = 8
This line is from parent (Pid 1615), value = 9
This line is from parent (Pid 1615), value = 10
This line is from parent (Pid 1615), value = 11
This line is from parent (Pid 1615), value = 12
This line is from parent (Pid 1615), value = 13
This line is from parent (Pid 1615), value = 14
This line is from parent (Pid 1615), value = 15
This line is from parent (Pid 1615), value = 16
This line is from parent (Pid 1615), value = 17
This line is from parent (Pid 1615), value = 18
This line is from parent (Pid 1615), value = 19
This line is from parent (Pid 1615), value = 20
*** Parent is done ***
denis.manley@apollo:~/OS2/week6$
```

Change position of sleep command and observe the effect!!!!

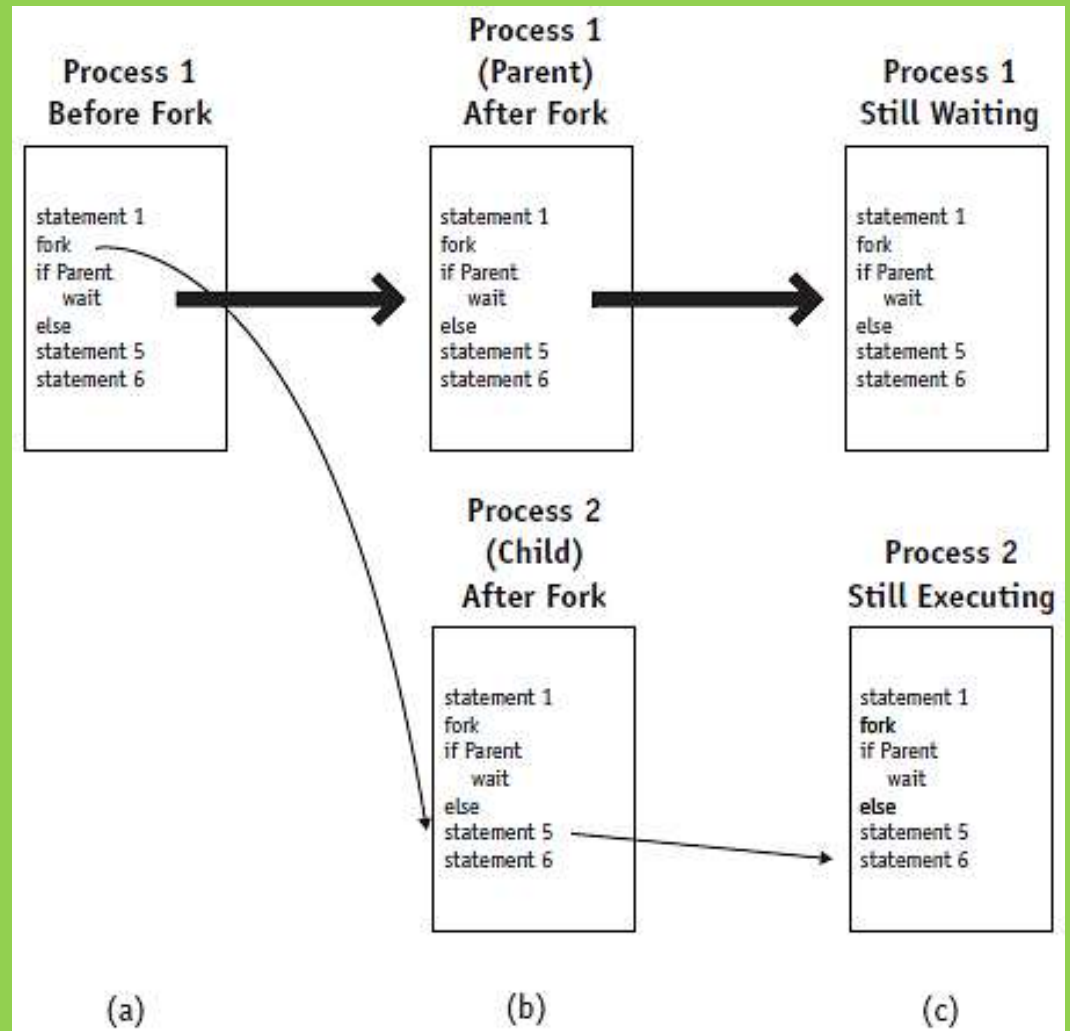
Synchronisation: Wait()

- **Wait()**
 - A better way to ensure “synchronisation is using the “wait()” command.
 - UNIX command: synchronizes process execution
 - Suspends parent until child finished
 - Program IF-THEN-ELSE structure
 - Controlled by pid value
 - pid > zero: parent process
 - pid = zero: child process
 - pid < zero: error in `fork` call
 - *Modify fork2.c program to use wait instead of sleep()*

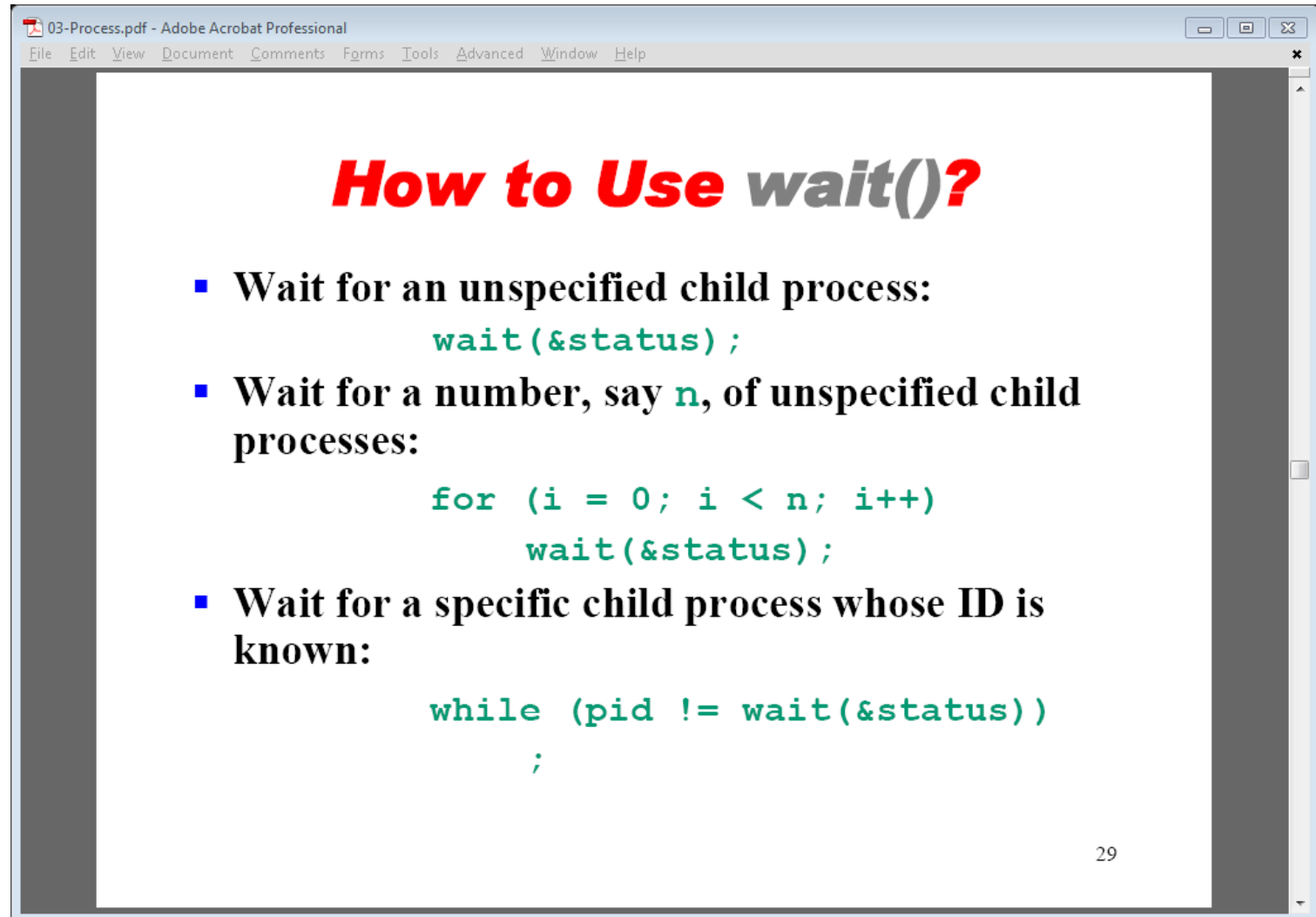
The wait command

The wait command used in conjunction with the fork command will synchronize the parent and child processes. In (a) the parent process is shown before the fork, (b) shows the parent and child after the fork, and (c) shows the parent and child during the wait.

© Cengage Learning 2014



Using a wait()



03-Process.pdf - Adobe Acrobat Professional

File Edit View Document Comments Forms Tools Advanced Window Help

How to Use wait()?

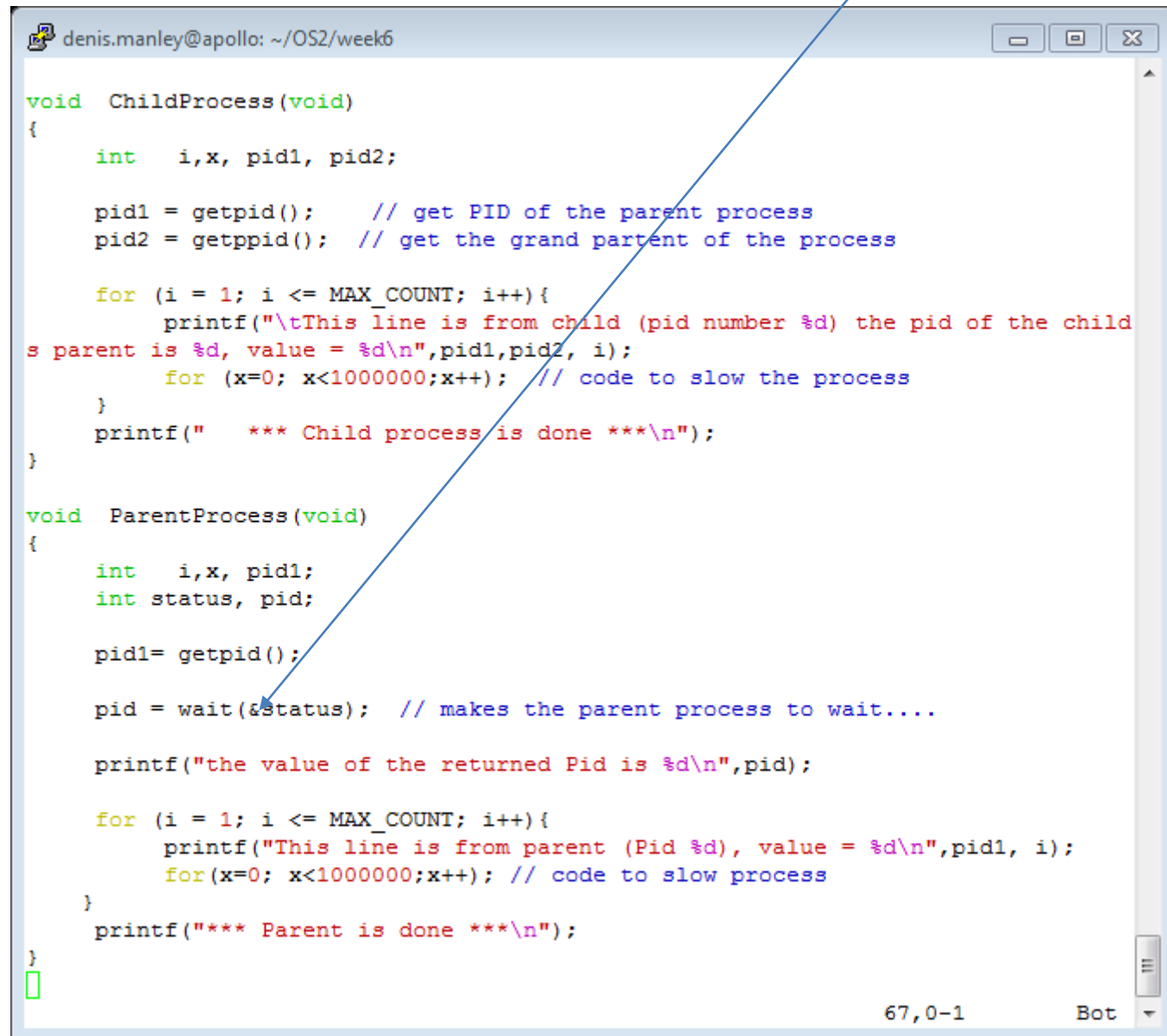
- Wait for an unspecified child process:
`wait(&status);`
- Wait for a number, say `n`, of unspecified child processes:

```
for (i = 0; i < n; i++)  
    wait(&status);
```
- Wait for a specific child process whose ID is known:

```
while (pid != wait(&status))  
    ;
```

29

Sample code Fork3.c (Wait command)



```
denis.manley@apollo: ~/OS2/week6

void ChildProcess(void)
{
    int i,x, pid1, pid2;

    pid1 = getpid(); // get PID of the parent process
    pid2 = getppid(); // get the grand partent of the process

    for (i = 1; i <= MAX_COUNT; i++){
        printf("\tThis line is from child (pid number %d) the pid of the child
s parent is %d, value = %d\n",pid1,pid2, i);
        for (x=0; x<1000000;x++); // code to slow the process
    }
    printf("    *** Child process is done ***\n");
}

void ParentProcess(void)
{
    int i,x, pid1;
    int status, pid;

    pid1= getpid();

    pid = wait(&status); // makes the parent process to wait....

    printf("the value of the returned Pid is %d\n",pid);

    for (i = 1; i <= MAX_COUNT; i++){
        printf("This line is from parent (Pid %d), value = %d\n",pid1, i);
        for(x=0; x<1000000;x++); // code to slow process
    }
    printf("**** Parent is done ***\n");
}

█
```

67,0-1 Bot

Execution of Fork3.c (*MAX_COUNT* = 5)

```
denis.manley@apollo: ~/OS2/week6
denis.manley@apollo:~/OS2/week6$ ./Fork3
the fork command returns the PID of the child: 4595
    This line is from child (pid number 4595) the pid of the childs parent is 4594, value = 1
    This line is from child (pid number 4595) the pid of the childs parent is 4594, value = 2
    This line is from child (pid number 4595) the pid of the childs parent is 4594, value = 3
    This line is from child (pid number 4595) the pid of the childs parent is 4594, value = 4
    This line is from child (pid number 4595) the pid of the childs parent is 4594, value = 5
    *** Child process is done ***

the value of the returned Pid is 4595

This line is from parent (Pid 4594), value = 1
This line is from parent (Pid 4594), value = 2
This line is from parent (Pid 4594), value = 3
This line is from parent (Pid 4594), value = 4
This line is from parent (Pid 4594), value = 5
*** Parent is done ***
denis.manley@apollo:~/OS2/week6$
```

int main(int argc, char **argv)

- argc is a variable that contains the number of command line arguments
- Argv is a character pointer array containing all the command line arguments.
- argv[0] is the first command line argument
- argv[1] is the second command line argument (if there is one...
- If the command line (after the prompt) is:
 - cp file1 file2
- what is the value returned by argv[2]?
- What value is returned by argc?

Pass command line arguments to c program: Fork4.c

```
denis.manley@apollo: ~/OS2/week6
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
    printf("--beginning of program\n");

    if (argc != 3) {
        printf("number of command line arguments should be 3\n");
        printf("exiting program.....\n");
        exit(0);
    }

    int counter = 0;
    int second;

    pid_t pid = fork();

    if (pid == 0)
    {
        // child process

        int max_c = atoi(argv[1]);

        printf("the second argument is %s\n", argv[1]);
        int i, x;
        for (i = 0; i < 5; ++i)
        {
            printf("child process: counter=%d\n", ++counter);
            for (x=0; x<max_c; x++);
        }
    }
    else if (pid > 0)
    {
        // parent process
        int j, x, status;

        int max_p = atoi(argv[2]);

        printf("the third argument is %s\n", argv[2]);
        //pid = wait(&status);
        for (j=0; j < 5; ++j)
        {
            printf("parent process: counter=%d\n", ++counter);
            for (x=0; x<max_p; x++);
        }
    }
}
```

```
denis.manley@soc-apollo: ~/OS2/week6
denis.manley@soc-apollo:~/OS2/week6$ ./fork4 1000 10000
--beginning of program
the third argument is 10000
parent process: counter=1
parent process: counter=2
the second argument is 1000
parent process: counter=3
child process: counter=1
child process: counter=2
child process: counter=3
child process: counter=4
child process: counter=5
--end of program--
parent process: counter=4
parent process: counter=5
--end of program--
denis.manley@soc-apollo:~/OS2/week6$ ./fork4 100000 1000000
--beginning of program
the third argument is 1000000
parent process: counter=1
the second argument is 100000
child process: counter=1
child process: counter=2
child process: counter=3
child process: counter=4
child process: counter=5
--end of program--
parent process: counter=2
parent process: counter=3
parent process: counter=4
parent process: counter=5
--end of program--
denis.manley@soc-apollo:~/OS2/week6$ ./fork4 10000000 100000000
--beginning of program
the third argument is 100000000
parent process: counter=1
the second argument is 10000000
child process: counter=1
child process: counter=2
child process: counter=3
child process: counter=4
child process: counter=5
--end of program--
parent process: counter=2
parent process: counter=3
parent process: counter=4
parent process: counter=5
--end of program--
denis.manley@soc-apollo:~/OS2/week6$
```

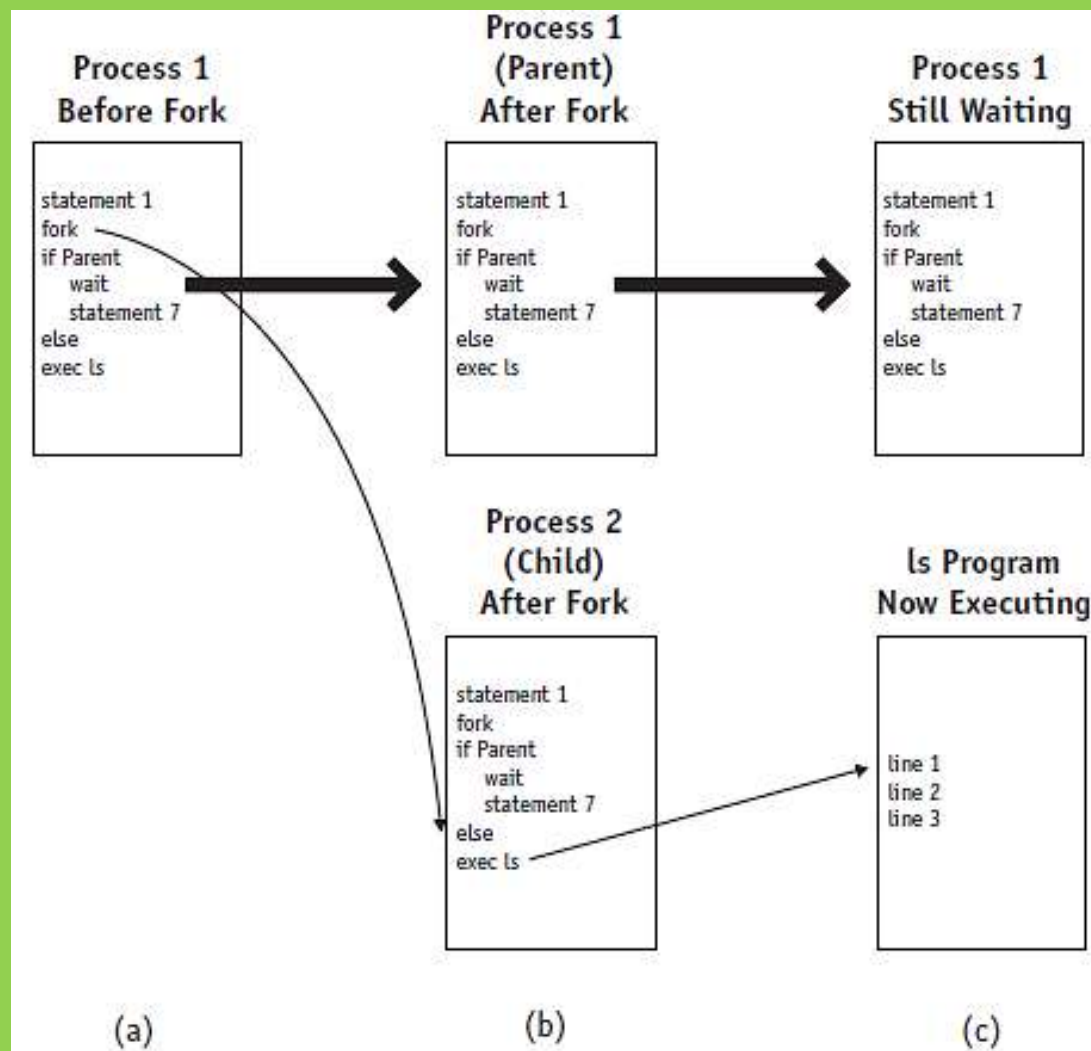
Process Creation: Unix

- How do we actually start a new program?
 - Use `exec` ((see *man exec* for more details)
 - Start new program execution from another program
- **E.g. `int execvp(char *prog, char *argv[])`**
 - Stops the current process (e.g. child of prompt)
 - Loads the program “prog” into the process’ address space
 - Initializes hardware context and args for the new program
 - Places the PCB of it onto the ready queue

Synchronization (cont'd.)

The exec command is used, mostly, after the fork and wait combination. In (a) the parent is shown before the fork, (b) shows the parent and child after the fork, and (c) shows how the child process (Process 2) is overlaid by the ls program after the exec command.

© Cengage Learning 2014



The exec commands

- The `exec()` system calls can be used after a `fork()` to replace the child process' memory space with a new program
- Successful `exec` call
 - Overlay second program over first
 - Only second program in memory
- No return from successful `exec` call
 - Parent-child concept: does not hold
- Create parent-child relationship
 - Call `fork`, `wait`, and `exec` in that order

Example with fork(), exec() and wait()

03-Process.pdf - Adobe Acrobat Professional

File Edit View Document Comments Forms Tools Advanced Window Help

execvp() : An Example 2/2

```
if ((pid = fork()) < 0) {  
    printf("fork() failed\n");  
    exit(1);  
}  
else if (pid == 0)  
    if (execvp(prog, argv) < 0) {  
        printf("execvp() failed\n");  
        exit(1);  
    }  
else  
    wait(&status);  
}
```

Diagram illustrating the execution of `execvp()` with the following components:

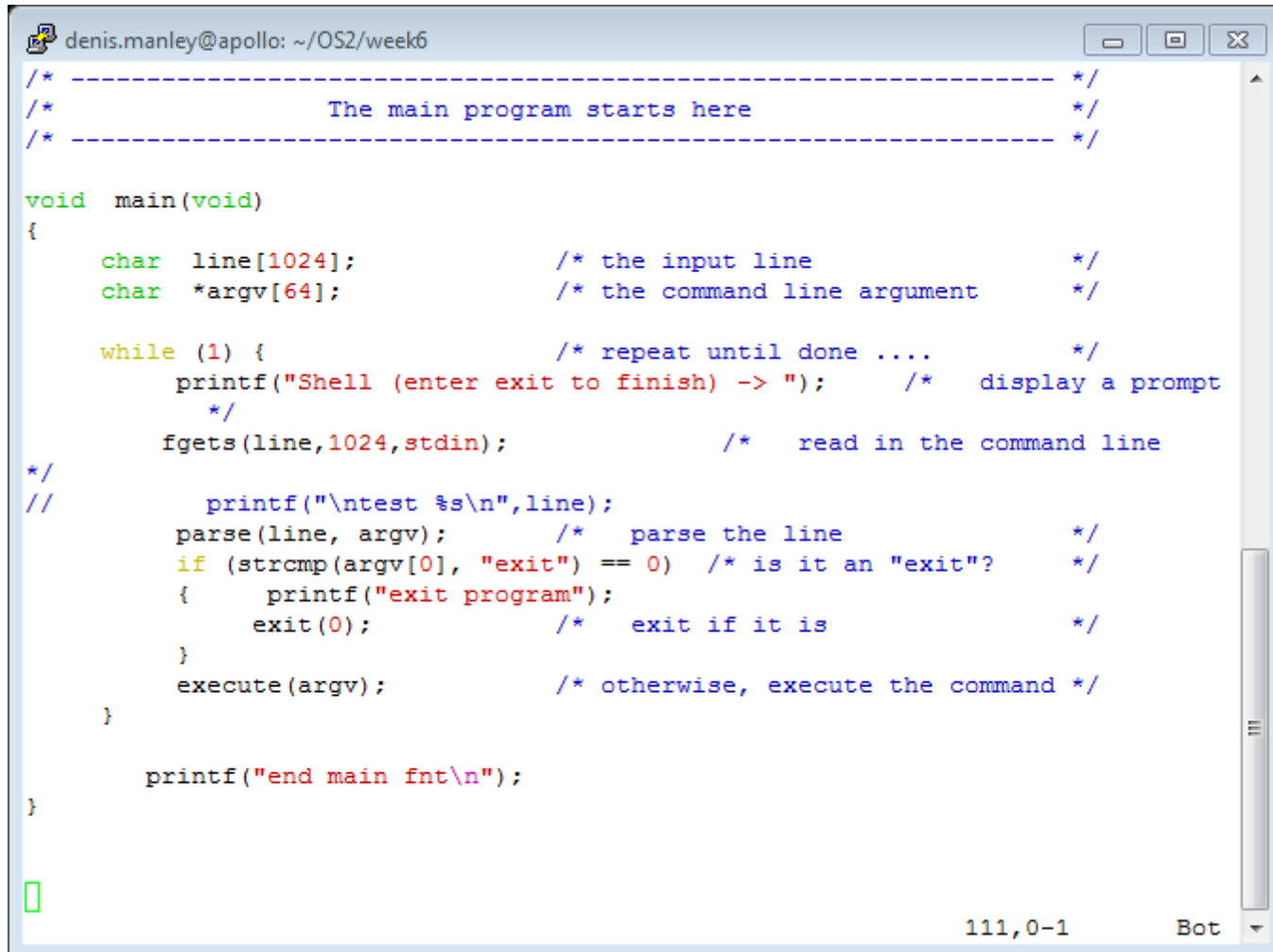
- `argv[]`: Array of arguments, shown as `["", "", "", "\0"]`.
- `prog`: Program name, shown as `["c", "p", "\0"]`.
- `argv[1]`: First argument, shown as `["t", "h", "i", "s", ".", "c", "\0"]`.
- `argv[2]`: Second argument, shown as `["t", "h", "a", "t", ".", "c", "\0"]`.

A red box labeled **execute program cp** points to the `prog` array, indicating the program to be executed.

33

Refer to **execvp.c** program; This program simulates a **linux command line**.

Main () for execvp program



```
denis.manley@apollo: ~/OS2/week6
/* ----- */
/*           The main program starts here           */
/* ----- */

void main(void)
{
    char line[1024];          /* the input line          */
    char *argv[64];          /* the command line argument */

    while (1) {              /* repeat until done .... */
        printf("Shell (enter exit to finish) -> "); /* display a prompt */
        /*
        fgets(line,1024,stdin);          /* read in the command line
    */
    //    printf("\ntest %s\n",line);
        parse(line, argv);          /* parse the line          */
        if (strcmp(argv[0], "exit") == 0) /* is it an "exit"?      */
        {
            printf("exit program");
            exit(0);          /* exit if it is          */
        }
        execute(argv);          /* otherwise, execute the command */
    }

    printf("end main fnt\n");
}

111,0-1 Bot
```

Parse function

```
denis.manley@apollo: ~/OS2/week6
/* ----- */
/* PROGRAM  shell.c
/*   This program reads in an input line, parses the input line
/*   into tokens, and use execvp() to execute the command.
/*
/*
/*   sample input ./exec  (run an exe file in current directory)
/*
/* ----- */

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>

/* ----- */
/* FUNCTION  parse:
/*   This function takes an input line and parse it into tokens.
/*   It first replaces all white spaces with zeros until it hits a
/*   non-white space character which indicates the beginning of an
/*   argument.  It saves the address to argv[], and then skips all
/*   non-white spaces which constitute the argument.
/* ----- */

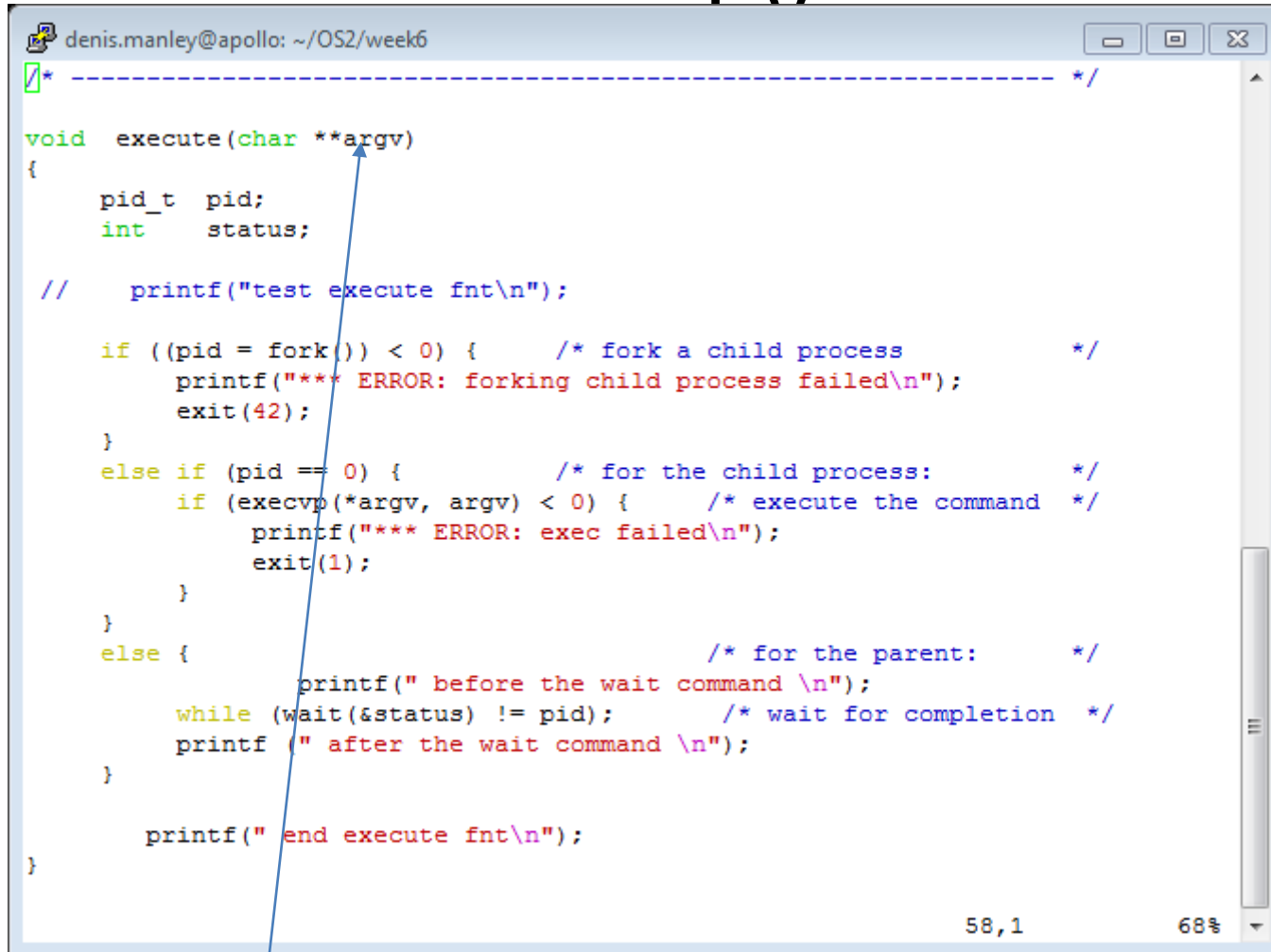
void parse(char *line, char **argv)
{
    //    printf("test parse...\n");

    while (*line != '\0') {
        while (*line == ' ' || *line == '\t' || *line == '\n')
            *line++ = '\0';
        *argv++ = line;
        while (*line != '\0' && *line != ' ' &&
               *line != '\t' && *line != '\n')
            line++;
    }

    *argv = '\0';
    //    printf("end parse\n");
}

1,1      Top
```

Execute function: equivalent to `execvp()`



```
denis.manley@apollo: ~/OS2/week6
/* ----- */

void execute(char **argv)
{
    pid_t pid;
    int status;

    // printf("test execute fnt\n");

    if ((pid = fork()) < 0) { /* fork a child process */
        printf("*** ERROR: forking child process failed\n");
        exit(42);
    }
    else if (pid == 0) { /* for the child process: */
        if (execvp(*argv, argv) < 0) { /* execute the command */
            printf("*** ERROR: exec failed\n");
            exit(1);
        }
    }
    else { /* for the parent: */
        printf(" before the wait command \n");
        while (wait(&status) != pid); /* wait for completion */
        printf(" after the wait command \n");
    }

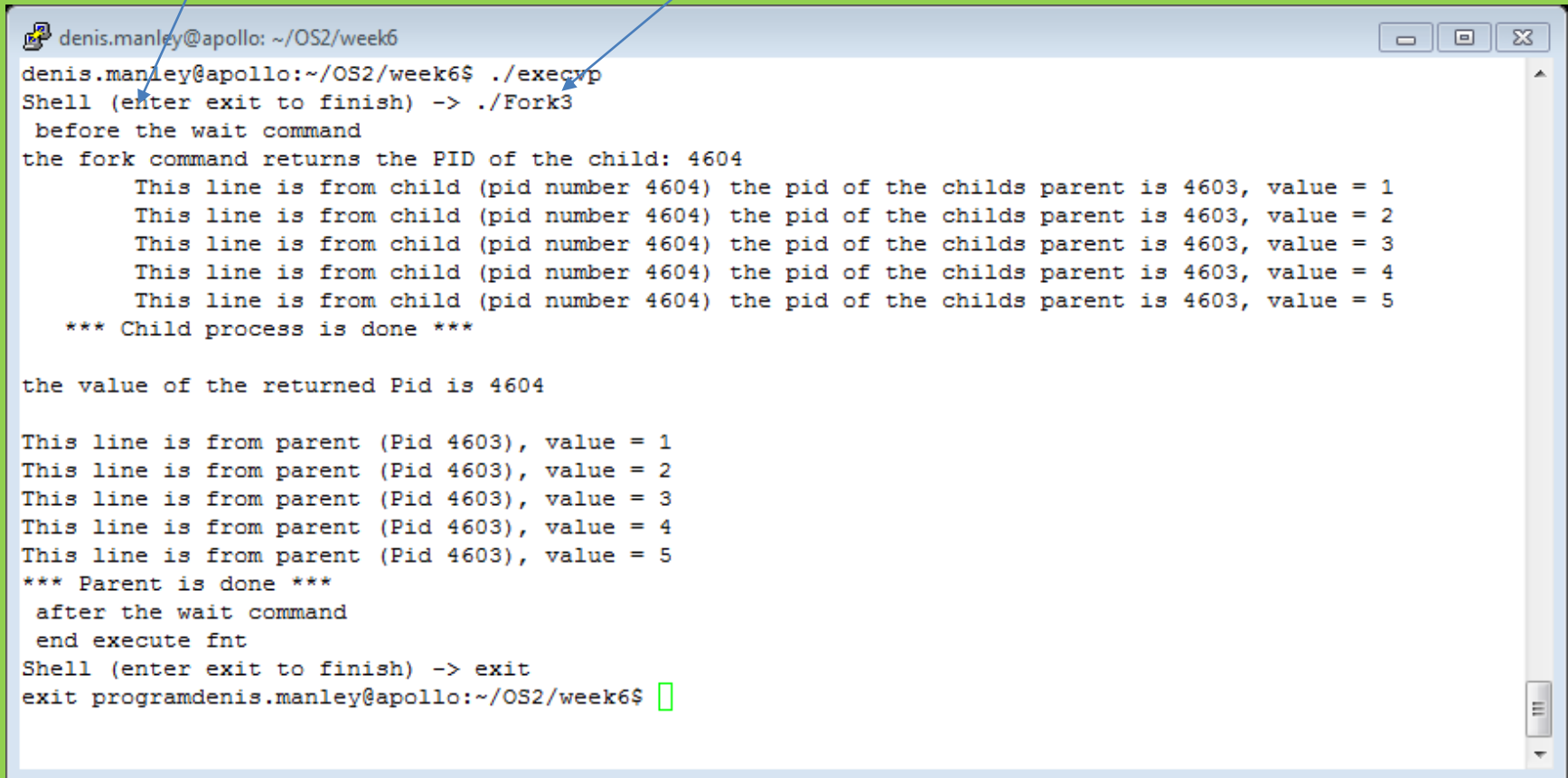
    printf(" end execute fnt\n");
}

58,1 68%
```

argv: An array of strings implemented a array of `char*` : `argv` is
**

Sample output execvp.c: it uses exec() command to run the program
./Fork3.

“Shell (enter exit to finish) is equivalent to the “linux prompt”.



```
denis.manley@apollo: ~/OS2/week6
denis.manley@apollo:~/OS2/week6$ ./execvp
Shell (enter exit to finish) -> ./Fork3
before the wait command
the fork command returns the PID of the child: 4604
    This line is from child (pid number 4604) the pid of the childs parent is 4603, value = 1
    This line is from child (pid number 4604) the pid of the childs parent is 4603, value = 2
    This line is from child (pid number 4604) the pid of the childs parent is 4603, value = 3
    This line is from child (pid number 4604) the pid of the childs parent is 4603, value = 4
    This line is from child (pid number 4604) the pid of the childs parent is 4603, value = 5
*** Child process is done ***

the value of the returned Pid is 4604

This line is from parent (Pid 4603), value = 1
This line is from parent (Pid 4603), value = 2
This line is from parent (Pid 4603), value = 3
This line is from parent (Pid 4603), value = 4
This line is from parent (Pid 4603), value = 5
*** Parent is done ***
after the wait command
end execute fnt
Shell (enter exit to finish) -> exit
exit programdenis.manley@apollo:~/OS2/week6$
```

remove **wait** command and see what happened: explain what is happening

Function heading

- In the `execvp` function we have:
 - `int execvp(char *prog, char *argv[])`
- These should not be confused with the
 - `int main (int argc, char *argv[]).`
 - The arguments in main refer to command line arguments:
 - `argc` is the number of command line arguments
 - `Argv` is an array of strings corresponding to the command line arguments: n.b. The program name is also a command line argument

Homework Exercise

- Compile and Run the program fork1.c fork2.c fork3.c and fork4.c **using the execvp program**
- Write a program that calls the fork command twice and each child will run a different functions e.g. child one calculates the cube of a command line argument and child two calculates the factorial of the 3rd commandline argument. You will need to use the wait command to synchronise output.: refer to fork3.c wait command