Operating System Practical exam

Subject Code : V20PC102

1. Write the steps for installing windows in a virtual machine / Workstation.

Answer 1: Some of the most popular virtual machine software are Hyper-V, VirtualBox, and VMware Workstation Player. You can download them from their official websites or use the links below:

* [Hyper-V](https://learn.microsoft.com/en-us/virtualization/hyper-v-on-windows/quick-start/create-virtual-machine)
* [VirtualBox](https://www.extremetech.com/computing/198427-how-to-install-windows-10-in-a-virtual-machine)
* [VMware Workstation Player]

The steps for installing Windows in a virtual machine are similar for each software, but they may have some differences in the interface and options. Here is a general overview of the steps:

1. Download the Windows ISO file from the [Windows 10 download page](https://www.petri.com/how-to-install-windows-10-in-a-virtual-machine) or use the Media Creation Tool to create one.
2. Launch the virtual machine software and create a new virtual machine. Give it a name and choose a location for the files.
3. Select the generation or type of the virtual machine. For Windows 10, you should choose Generation 2 or Microsoft Windows.
4. Allocate RAM and disk space for the virtual machine. The minimum requirements for Windows 10 are 2 GB of RAM and 20 GB of disk space, but you can increase them if you have enough resources on your host machine.
5. Select a virtual network adapter for the virtual machine. This will allow it to connect to the internet and your local network.
6. Locate the Windows ISO file and attach it to the virtual machine as a bootable image file.
7. Start the virtual machine and follow the instructions to install Windows 10 on it.
8. Install any additional drivers or tools that are required by the virtual machine software, such as VirtualBox Guest Additions or VMware Tools.

2. Write the steps for Installing Ubuntu/Linux in a virtual machine/workstation.

Answer 2. There are different ways to install Ubuntu/Linux in a virtual machine, depending on the software you use. Some of the most popular virtual machine software are Hyper-V, VirtualBox, and VMware Workstation Player. You can download them from their official websites or use the links below:

* [Hyper-V](https://www.howtogeek.com/796988/how-to-install-linux-in-virtualbox/)
* [VirtualBox](https://apporbit.com/how-to-install-ubuntu-in-vmware/)
* [VMware Workstation Player](https://www.makeuseof.com/install-ubuntu-on-vmware-workstation/)

The steps for installing Ubuntu/Linux in a virtual machine are similar for each software, but they may have some differences in the interface and options. Here is a general overview of the steps:

1. Download the Ubuntu ISO file from the [Ubuntu download page] or use the Rufus tool to create one.
2. Launch the virtual machine software and create a new virtual machine. Give it a name and choose a location for the files.
3. Select the type of the virtual machine. For Ubuntu/Linux, you should choose Linux and then select the specific distribution and version you want to install.
4. Allocate RAM and disk space for the virtual machine. The minimum requirements for Ubuntu are 2 GB of RAM and 25 GB of disk space, but you can increase them if you have enough resources on your host machine.
5. Select a virtual network adapter for the virtual machine. This will allow it to connect to the internet and your local network.
6. Locate the Ubuntu ISO file and attach it to the virtual machine as a bootable image file.
7. Start the virtual machine and follow the instructions to install Ubuntu/Linux on it.
8. Install any additional drivers or tools that are required by the virtual machine software, such as VirtualBox Guest Additions or VMware Tools.

3. Create a process using fork () system call and use getpid(), getppid() functions along with wait() and exit () using C/C++ programming.

To create a process using fork () system call and use getpid (), getppid (), wait () and exit () functions in C/C++ programming, you can follow these steps:

1. Include the necessary header files, such as stdio.h, stdlib.h, sys/types.h, and unistd.h.
2. Declare a variable to store the process ID returned by fork (), such as pid\_t pid.
3. Call fork () and assign its return value to pid. This will create a child process that is a duplicate of the parent process, except for the pid value.
4. Use an if-else statement to check the value of pid and perform different actions for the parent and child processes.
5. In the child process (pid == 0), you can print the process ID of the child and the parent using getpid () and getppid () functions, respectively. You can also perform any other tasks that you want the child to do, such as executing another program using exec () function or generating some output. When the child finishes its work, you can call exit () function with an argument that indicates the exit status of the child, such as 0 for success or 1 for failure.
6. In the parent process (pid > 0), you can print the process ID of the parent and the child using getpid () and pid, respectively. You can also perform any other tasks that you want the parent to do, such as waiting for the child to finish using wait () function. The wait () function takes a pointer to an integer variable as an argument and blocks the parent until the child terminates or a signal is received. The wait () function also returns the process ID of the terminated child and stores its exit status in the integer variable pointed by its argument. You can use WIFEXITED and WEXITSTATUS macros to check whether the child exited normally and what was its exit status, respectively. You can also print some messages based on these information.
7. In case of an error (pid < 0), you can print an error message using perror () function and exit with a non-zero status using exit () function.

Here is an example code that demonstrates these steps:

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

// Declare a variable to store the process ID

pid\_t pid;

// Call fork() and assign its return value to pid

pid = fork();

// Check the value of pid

if (pid == 0) // Child process

{

// Print the process ID of the child and the parent

printf("Child process ID: %d\n", getpid());

printf("Parent process ID: %d\n", getppid());

// Perform any other tasks for the child

// For example, execute another program using exec() function

// char \*args[] = {"ls", "-l", NULL};

// execvp(args[0], args);

// Exit with a status of 0 (success)

exit(0);

}

else if (pid > 0) // Parent process

{

// Declare a variable to store the exit status of the child

int status;

// Print the process ID of the parent and the child

printf("Parent process ID: %d\n", getpid());

printf("Child process ID: %d\n", pid);

// Perform any other tasks for the parent

// For example, wait for the child to finish using wait() function

wait(&status);

// Check whether the child exited normally and what was its exit status

if (WIFEXITED(status))

{

printf("Child exited normally with status %d\n", WEXITSTATUS(status));

}

else

{

printf("Child exited abnormally\n");

}

}

else // Error

{

// Print an error message using perror() function

perror("fork");

// Exit with a non-zero status

exit(1);

}

return 0;

}

4. Write a program to implement a unidirectional pipe under IPC using C/C++ programming.

To write a program to implement a unidirectional pipe under IPC using C/C++ programming, you can follow these steps:

1. Include the necessary header files, such as stdio.h, stdlib.h, sys/types.h, and unistd.h.
2. Declare an array of two integers to store the file descriptors for the pipe, such as int pipefd[2](https://stackoverflow.com/questions/42431253/unidirectional-pipes-in-c).
3. Call the pipe() function with the array as an argument to create a unidirectional pipe. The pipe() function returns 0 on success and -1 on error. The array will contain the file descriptors for the reading end (pipefd[0]) and the writing end (pipefd[1](https://www.geeksforgeeks.org/ipc-technique-pipes/)) of the pipe.
4. Call the fork() function to create a child process that is a duplicate of the parent process. The fork() function returns the process ID of the child to the parent, 0 to the child, and -1 on error.
5. Use an if-else statement to check the return value of fork() and perform different actions for the parent and child processes.
6. In the parent process, close the reading end of the pipe (pipefd[0]) and write some data to the writing end of the pipe (pipefd[1](https://www.geeksforgeeks.org/ipc-technique-pipes/)) using the write() function. You can use any data type or format you want, such as a string, a number, or a structure. You can also use a loop to write multiple data items to the pipe. After writing, close the writing end of the pipe (pipefd[1](https://www.geeksforgeeks.org/ipc-technique-pipes/)) and exit with a status of 0.
7. In the child process, close the writing end of the pipe (pipefd[1](https://www.geeksforgeeks.org/ipc-technique-pipes/)) and read some data from the reading end of the pipe (pipefd[0]) using the read() function. You should use the same data type or format as the parent process, and allocate enough memory to store the data. You can also use a loop to read multiple data items from the pipe. After reading, print or process the data as you wish, and close the reading end of the pipe (pipefd[0]) and exit with a status of 0.
8. In case of an error in either pipe() or fork(), print an error message using perror() function and exit with a non-zero status.

Here is an example code that demonstrates these steps:

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

// Declare an array of two integers to store the file descriptors for the pipe

int pipefd[2];

// Call pipe() function with the array as an argument to create a unidirectional pipe

if (pipe(pipefd) == -1)

{

// Print an error message and exit with a non-zero status in case of an error

perror("pipe");

exit(1);

}

// Call fork() function to create a child process

pid\_t pid = fork();

// Check the return value of fork()

if (pid > 0) // Parent process

{

// Close the reading end of the pipe

close(pipefd[0]);

// Write some data to the writing end of the pipe

char \*msg = "Hello from parent";

write(pipefd[1], msg, sizeof(msg));

// Close the writing end of the pipe

close(pipefd[1]);

// Exit with a status of 0

exit(0);

}

else if (pid == 0) // Child process

{

// Close the writing end of the pipe

close(pipefd[1]);

// Read some data from the reading end of the pipe

char buf[20];

read(pipefd[0], buf, sizeof(buf));

// Print or process the data as you wish

printf("Child received: %s\n", buf);

// Close the reading end of the pipe

close(pipefd[0]);

// Exit with a status of 0

exit(0);

}

else // Error

{

// Print an error message and exit with a non-zero status in case of an error

perror("fork");

exit(1);

}

return 0;

}

4. write a program to implement a unidirectional pipe under IPC using C/C++ programming.

Answer 4.