**ASSIGNMENT 4**

**1. Process Management System calls fork(), exec(), wait(), exit().**

* **Process Management System Calls**

Process management system calls are used to create, terminate, and manage processes in Linux.

**1.1. fork()**

The fork() system call is used to create a new process by duplicating the calling process. The new process is called the child process, and it gets a unique process ID (PID). The child process gets a copy of the parent's memory space.

**Syntax:**

pid\_t fork(void);

**Return Value:**

* On success, it returns the child process ID to the parent and returns 0 to the child.
* On failure, it returns -1 and sets the errno variable.

**Example:**

#include <stdio.h>

#include <unistd.h>

int main() {

pid\_t pid = fork();

if (pid == 0) {

printf("Child Process\n");

} else {

printf("Parent Process\n");

}

return 0;

}

**1.2. exec()**

The exec() family of system calls replaces the current process with a new process. It loads the program into the process's memory and runs it. The most commonly used version is execl(), but there are other versions like execp(), execv(), etc.

**Syntax:**

int execvp(const char \*file, char \*const argv[]);

**Example:**

#include <stdio.h>

#include <unistd.h>

int main() {

execlp("/bin/ls", "ls", NULL);

printf("This will not be printed\n");

return 0;

}

**1.3. wait()**

The wait() system call is used by a process to wait for the termination of one of its child processes. It returns the PID of the terminated child process.

**Syntax:**

pid\_t wait(int \*status);

**Example:**

#include <stdio.h>

#include <sys/wait.h>

#include <unistd.h>

int main() {

pid\_t pid = fork();

if (pid == 0) {

// Child process

printf("Child Process\n");

} else {

// Parent process

wait(NULL);

printf("Parent Process: Child terminated\n");

}

return 0;

}

**1.4. exit()**

The exit() system call terminates the calling process and returns an exit status to the parent process.

**Syntax:**

void exit(int status);

**Example:**

#include <stdio.h>

#include <stdlib.h>

int main() {

printf("This program will exit now.\n");

exit(0);

}

**2. File Management System calls open(), read(), write(), close().**

* **File Management System Calls**

File management system calls are used for performing file-related operations like opening, reading, writing, and closing files.

**2.1. open()**

The open() system call is used to open a file and return a file descriptor.

**Syntax:**

int open(const char \*pathname, int flags, mode\_t mode);

**Example:**

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

int main() {

int fd = open("test.txt", O\_CREAT | O\_WRONLY, 0644);

if (fd == -1) {

perror("Error opening file");

return -1;

}

close(fd);

return 0;

}

**2.2. read()**

The read() system call reads data from a file descriptor into a buffer.

**Syntax:**

ssize\_t read(int fd, void \*buf, size\_t count);

**Example:**

#include <stdio.h>

#include <unistd.h>

#include <fcntl.h>

int main() {

int fd = open("test.txt", O\_RDONLY);

char buffer[100];

if (fd == -1) {

perror("Error opening file");

return -1;

}

read(fd, buffer, sizeof(buffer));

printf("Data from file: %s\n", buffer);

close(fd);

return 0;

}

**2.3. write()**

The write() system call writes data to a file descriptor.

**Syntax:**

ssize\_t write(int fd, const void \*buf, size\_t count);

**Example:**

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

int main() {

int fd = open("test.txt", O\_WRONLY);

if (fd == -1) {

perror("Error opening file");

return -1;

}

write(fd, "Hello, World!", 13);

close(fd);

return 0;

}

**2.4. close()**

The close() system call closes a file descriptor.

**Syntax:**

int close(int fd);

**Example:**

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

int main() {

int fd = open("test.txt", O\_CREAT | O\_WRONLY, 0644);

if (fd == -1) {

perror("Error opening file");

return -1;

}

close(fd);

return 0;

}

**3. Device Management System calls read(), write(), ioctl(), select().**

* **Device Management System Calls**

Device management system calls manage devices, such as reading from and writing to hardware devices.

**3.1. ioctl()**

The ioctl() system call controls devices by sending various control commands to a device.

**Syntax:**

int ioctl(int fd, unsigned long request, ...);

**Example:**

#include <stdio.h>

#include <sys/ioctl.h>

#include <unistd.h>

int main() {

int fd = open("/dev/sda", O\_RDWR);

if (fd == -1) {

perror("Error opening device");

return -1;

}

// Example: Get device size using ioctl (this is just an example)

ioctl(fd, BLKGETSIZE, &size);

close(fd);

return 0;

}

**3.2. select()**

The select() system call is used to monitor multiple file descriptors to see if any are ready for I/O operations.

**Syntax:**

int select(int nfds, fd\_set \*readfds, fd\_set \*writefds, fd\_set \*exceptfds, struct timeval \*timeout);

**Example:**

#include <stdio.h>

#include <sys/select.h>

#include <unistd.h>

int main() {

fd\_set readfds;

struct timeval timeout;

FD\_ZERO(&readfds);

FD\_SET(0, &readfds); // Monitor stdin

timeout.tv\_sec = 5;

timeout.tv\_usec = 0;

int ret = select(1, &readfds, NULL, NULL, &timeout);

if (ret == 0) {

printf("Timeout occurred! No data input.\n");

} else {

printf("Data is available for reading.\n");

}

return 0;

}

**4. Network Management System calls socket(), connect(), send(), recv().**

* **Network Management System Calls**

Network management system calls are used for socket programming to communicate over a network.

**4.1. socket()**

The socket() system call creates a new socket.

**Syntax:**

int socket(int domain, int type, int protocol);

**Example:**

#include <stdio.h>

#include <sys/socket.h>

#include <netinet/in.h>

int main() {

int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

if (sockfd == -1) {

perror("Socket creation failed");

return -1;

}

printf("Socket created successfully\n");

return 0;

}

**4.2. connect()**

The connect() system call establishes a connection to a specified socket.

**Syntax:**

int connect(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);

**Example:**

#include <stdio.h>

#include <sys/socket.h>

#include <netinet/in.h>

int main() {

int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

struct sockaddr\_in server\_addr;

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(8080);

server\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

connect(sockfd, (struct sockaddr \*)&server\_addr, sizeof(server\_addr));

printf("Connected to server\n");

return 0;

}

**4.3. send() and recv()**

The send() system call sends data to a socket, and recv() receives data from a socket.

**Syntax:**

ssize\_t send(int sockfd, const void \*buf, size\_t len, int flags);

ssize\_t recv(int sockfd, void \*buf, size\_t len, int flags);

**Example:**

#include <stdio.h>

#include <sys/socket.h>

int main() {

int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

char message[] = "Hello Server!";

send(sockfd, message, sizeof(message), 0);

recv(sockfd, message, sizeof(message), 0);

printf("Received message: %s\n", message);

return 0;

}

**5. System Information Management System calls getpid(), getuid(), gethostname(), sysinfo().**

* **System Information Management System Calls**

These system calls allow processes to query information about the system, such as process IDs and user IDs.

**5.1. getpid()**

The getpid() system call returns the process ID of the calling process.

**Syntax:**

pid\_t getpid(void);

**Example:**

#include <stdio.h>

#include <unistd.h>

int main() {

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

return 0;

}

**5.2. getuid()**

The getuid() system call returns the user ID of the calling process.

**Syntax:**

uid\_t getuid(void);

**Example:**

#include <stdio.h>

#include <unistd.h>

int main() {

printf("User ID: %d\n", getuid());

return 0;

}

**5.3. gethostname()**

The gethostname() system call retrieves the hostname of the system.

**Syntax:**

int gethostname(char \*name, size\_t len);

**Example:**

#include <stdio.h>

#include <unistd.h>

int main() {

char hostname[256];

gethostname(hostname, sizeof(hostname));

printf("Hostname: %s\n", hostname);

return 0;

}

**5.4. sysinfo()**

The sysinfo() system call provides system statistics like uptime, load average, and available memory.

**Syntax:**

int sysinfo(struct sysinfo \*info);

**Example:**

#include <stdio.h>

#include <sys/sysinfo.h>

int main() {

struct sysinfo info;

sysinfo(&info);

printf("System uptime: %ld seconds\n", info.uptime);

return 0;

}