1) WAP to demonstrate the usage of wait()

wait() is an essential mechanism for coordinating the execution of parent and child processes, managing system resources, and handling the termination of child processes in Unix-like operating systems.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

#include <sys/types.h>

int main() {

pid\_t pid;

int status;

pid = fork();

if (pid < 0) {

perror("fork");

exit(EXIT\_FAILURE);

} else if (pid == 0) {

// Child process

printf("Child process executing...\n");

sleep(2); // Simulate some work

printf("Child process completed.\n");

exit(EXIT\_SUCCESS);

} else {

// Parent process

printf("Parent process waiting for child...\n");

wait(&status); // Wait for child to terminate

printf("Child process terminated.\n");

if (WIFEXITED(status)) {

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

} else if (WIFSIGNALED(status)) {

printf("Child process terminated by signal: %d\n", WTERMSIG(status));

}

}

return 0;

}

2) WAP to demonstrate the usage of waited() system call.

The waitid() system call is used to wait for the termination of a specific process or any child process. It provides more flexibility than wait() or waitpid() by allowing the caller to specify various options and receive detailed information about the terminated process.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

#include <sys/types.h>

int main() {

pid\_t pid;

int status;

siginfo\_t info;

pid = fork();

if (pid < 0) {

perror("fork");

exit(EXIT\_FAILURE);

} else if (pid == 0) {

// Child process

printf("Child process executing...\n");

sleep(2); // Simulate some work

printf("Child process completed.\n");

exit(EXIT\_SUCCESS);

} else {

// Parent process

printf("Parent process waiting for child...\n");

if (waitid(P\_PID, pid, &info, WEXITED | WSTOPPED | WCONTINUED) == -1) {

perror("waitid");

exit(EXIT\_FAILURE);

}

printf("Child process terminated.\n");

printf("Process ID of terminated child: %d\n", info.si\_pid);

printf("Exit status of terminated child: %d\n", info.si\_status);

}

return 0;

}

3) WAP to demonstrate the usage of waitpid() system call

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

#include <sys/types.h>

int main() {

pid\_t pid;

int status;

pid = fork();

if (pid < 0) {

perror("fork");

exit(EXIT\_FAILURE);

} else if (pid == 0) {

// Child process

printf("Child process executing...\n");

sleep(2); // Simulate some work

printf("Child process completed.\n");

exit(EXIT\_SUCCESS);

} else {

// Parent process

printf("Parent process waiting for child...\n");

if (waitpid(pid, &status, 0) == -1) {

perror("waitpid");

exit(EXIT\_FAILURE);

}

printf("Child process terminated.\n");

if (WIFEXITED(status)) {

printf("Exit status of terminated child: %d\n", WEXITSTATUS(status));

} else if (WIFSIGNALED(status)) {

printf("Terminated by signal: %d\n", WTERMSIG(status));

}

}

return 0;

}

4) WAP to demonstrate the usage of wait3() system call.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

#include <sys/types.h>

#include <sys/resource.h>

int main() {

pid\_t pid;

int status;

struct rusage usage;

pid = fork();

if (pid < 0) {

perror("fork");

exit(EXIT\_FAILURE);

} else if (pid == 0) {

// Child process

printf("Child process executing...\n");

sleep(2); // Simulate some work

printf("Child process completed.\n");

exit(EXIT\_SUCCESS);

} else {

// Parent process

printf("Parent process waiting for child...\n");

wait3(&status, 0, &usage); // Wait for child to terminate

printf("Child process terminated.\n");

printf("CPU time used by child: %ld.%06ld seconds\n", usage.ru\_utime.tv\_sec, usage.ru\_utime.tv\_usec);

printf("CPU time used by system on behalf of child: %ld.%06ld seconds\n", usage.ru\_stime.tv\_sec, usage.ru\_stime.tv\_usec);

}

return 0;

}

5) WAP to demonstrate the usage of wait4() system call

The wait4() system call is similar to wait3(), but it provides additional options for controlling how resource usage information is returned. A simple C program demonstrating the usage of wait4() is given below

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

#include <sys/types.h>

#include <sys/resource.h>

int main() {

pid\_t pid;

int status;

struct rusage usage;

pid = fork();

if (pid < 0) {

perror("fork");

exit(EXIT\_FAILURE);

} else if (pid == 0) {

// Child process

printf("Child process executing...\n");

sleep(2); // Simulate some work

printf("Child process completed.\n");

exit(EXIT\_SUCCESS);

} else {

// Parent process

printf("Parent process waiting for child...\n");

wait4(pid, &status, 0, &usage); // Wait for child to terminate

printf("Child process terminated.\n");

printf("CPU time used by child: %ld.%06ld seconds\n", usage.ru\_utime.tv\_sec, usage.ru\_utime.tv\_usec);

printf("CPU time used by system on behalf of child: %ld.%06ld seconds\n", usage.ru\_stime.tv\_sec, usage.ru\_stime.tv\_usec);

}

return 0;

}