Lab Sheet: Process Management in Windows 10 (C/C++)

# 1. Title

Study of Process Management in Windows 10 using C/C++

# 2. Objectives

- To understand the concept of processes in Windows operating system.

- To learn how to create and manage processes using Win32 API in C/C++.

- To observe process attributes such as Process ID (PID), Parent Process ID (PPID), and execution behavior.

- To practice writing programs that launch and synchronize processes.

# 3. Prerequisites

- Basic knowledge of C/C++ programming.

- Familiarity with Windows Operating System.

- Installed IDE/compiler supporting Win32 API (e.g., Visual Studio, Code::Blocks with MinGW).

# 4. Theory

- Process: An executing instance of a program.

- Windows API Functions for process management:

• CreateProcess() → creates a new process and its primary thread.

• GetCurrentProcessId() → retrieves the current process ID.

• GetCurrentThreadId() → retrieves the current thread ID.

• WaitForSingleObject() → waits for a process to finish execution.

• TerminateProcess() → terminates a process.

- Each process has attributes: Process ID, Handle, Exit Code, Priority, Security Attributes.

# 5. Lab Tasks

Task 1: Display Current Process Information

- Write a C program to print the Process ID and Thread ID.

Task 2: Create a New Process

- Write a C/C++ program using CreateProcess() to launch an application (e.g., Notepad).

- Print the Parent Process ID and the Child Process ID.

Task 3: Process Synchronization

- Extend Task 2: The parent process should wait until the child process terminates before continuing execution.

Task 4: Multiple Processes

- Modify the program to create two child processes concurrently and display their process IDs.

# 6. Solutions

See the example programs provided below.

# Solution for Task 2: Create Notepad Process

#include <windows.h>

#include <stdio.h>

int main() {

STARTUPINFO si;

PROCESS\_INFORMATION pi;

ZeroMemory(&si, sizeof(si));

si.cb = sizeof(si);

ZeroMemory(&pi, sizeof(pi));

printf("Parent Process ID: %lu\n", GetCurrentProcessId());

if (!CreateProcess(

"C:\\Windows\\System32\\notepad.exe",

NULL,

NULL,

NULL,

FALSE,

0,

NULL,

NULL,

&si,

&pi)

) {

printf("CreateProcess failed (%d).\n", GetLastError());

return 1;

}

printf("Child Process ID: %lu\n", pi.dwProcessId);

WaitForSingleObject(pi.hProcess, INFINITE);

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

CloseHandle(pi.hProcess);

CloseHandle(pi.hThread);

return 0;

}

# 7. Expected Output

Parent Process ID: 5678

Child Process ID: 1234

Child process finished.

# 8. Lab Questions

1. What is a process? How is it different from a program?

2. Which Windows API is used to create a process?

3. How can we make the parent wait for the child process?

4. What happens if you don’t call WaitForSingleObject() after CreateProcess()?

5. How do process IDs help in process management?

# 9. Result / Conclusion

- Students should be able to write programs that demonstrate process creation, execution, and termination in Windows 10.

- The mechanism of parent-child relationship in process management should be clearly understood.

# Solution for Task 1: Display Current Process Information

#include <windows.h>  
#include <stdio.h>  
  
int main() {  
 printf("Current Process ID: %lu\n", GetCurrentProcessId());  
 printf("Current Thread ID : %lu\n", GetCurrentThreadId());  
 return 0;  
}

# Solution for Task 3: Process Synchronization

#include <windows.h>  
#include <stdio.h>  
  
int main() {  
 STARTUPINFO si;  
 PROCESS\_INFORMATION pi;  
  
 ZeroMemory(&si, sizeof(si));  
 si.cb = sizeof(si);  
 ZeroMemory(&pi, sizeof(pi));  
  
 printf("Parent Process ID: %lu\n", GetCurrentProcessId());  
  
 if (!CreateProcess("C:\\Windows\\System32\\notepad.exe",  
 NULL, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi)) {  
 printf("CreateProcess failed (%d).\n", GetLastError());  
 return 1;  
 }  
  
 printf("Child Process ID: %lu\n", pi.dwProcessId);  
 printf("Waiting for child process to finish...\n");  
 WaitForSingleObject(pi.hProcess, INFINITE);  
 printf("Child process finished.\n");  
  
 CloseHandle(pi.hProcess);  
 CloseHandle(pi.hThread);  
 return 0;  
}

# Solution for Task 4: Multiple Processes

#include <windows.h>  
#include <stdio.h>  
  
int main() {  
 STARTUPINFO si1, si2;  
 PROCESS\_INFORMATION pi1, pi2;  
  
 ZeroMemory(&si1, sizeof(si1));  
 si1.cb = sizeof(si1);  
 ZeroMemory(&pi1, sizeof(pi1));  
  
 ZeroMemory(&si2, sizeof(si2));  
 si2.cb = sizeof(si2);  
 ZeroMemory(&pi2, sizeof(pi2));  
  
 printf("Parent Process ID: %lu\n", GetCurrentProcessId());  
  
 if (!CreateProcess("C:\\Windows\\System32\\notepad.exe", NULL, NULL, NULL, FALSE, 0, NULL, NULL, &si1, &pi1)) {  
 printf("First CreateProcess failed (%d).\n", GetLastError());  
 return 1;  
 }  
 printf("First Child Process ID: %lu\n", pi1.dwProcessId);  
  
 if (!CreateProcess("C:\\Windows\\System32\\mspaint.exe", NULL, NULL, NULL, FALSE, 0, NULL, NULL, &si2, &pi2)) {  
 printf("Second CreateProcess failed (%d).\n", GetLastError());  
 return 1;  
 }  
 printf("Second Child Process ID: %lu\n", pi2.dwProcessId);  
  
 WaitForSingleObject(pi1.hProcess, INFINITE);  
 WaitForSingleObject(pi2.hProcess, INFINITE);  
  
 printf("Both child processes finished.\n");  
  
 CloseHandle(pi1.hProcess);  
 CloseHandle(pi1.hThread);  
 CloseHandle(pi2.hProcess);  
 CloseHandle(pi2.hThread);  
 return 0;  
}

# 9. Result / Conclusion

- Students should be able to write programs that demonstrate process creation, execution, and termination in Windows 10.  
- The mechanism of parent-child relationship in process management should be clearly understood.