MS SERVER 2016 PROCESSES AND THREADS MANGMENT

Asaad Qasim Shareef
163109042

Introduction

- Windows Server has powered a generation of organizations, from small businesses to large enterprises. No matter what your role in IT, you can be guaranteed you that have touched Windows. The mechanics of performing a live migration are the same as they were in previous versions of.

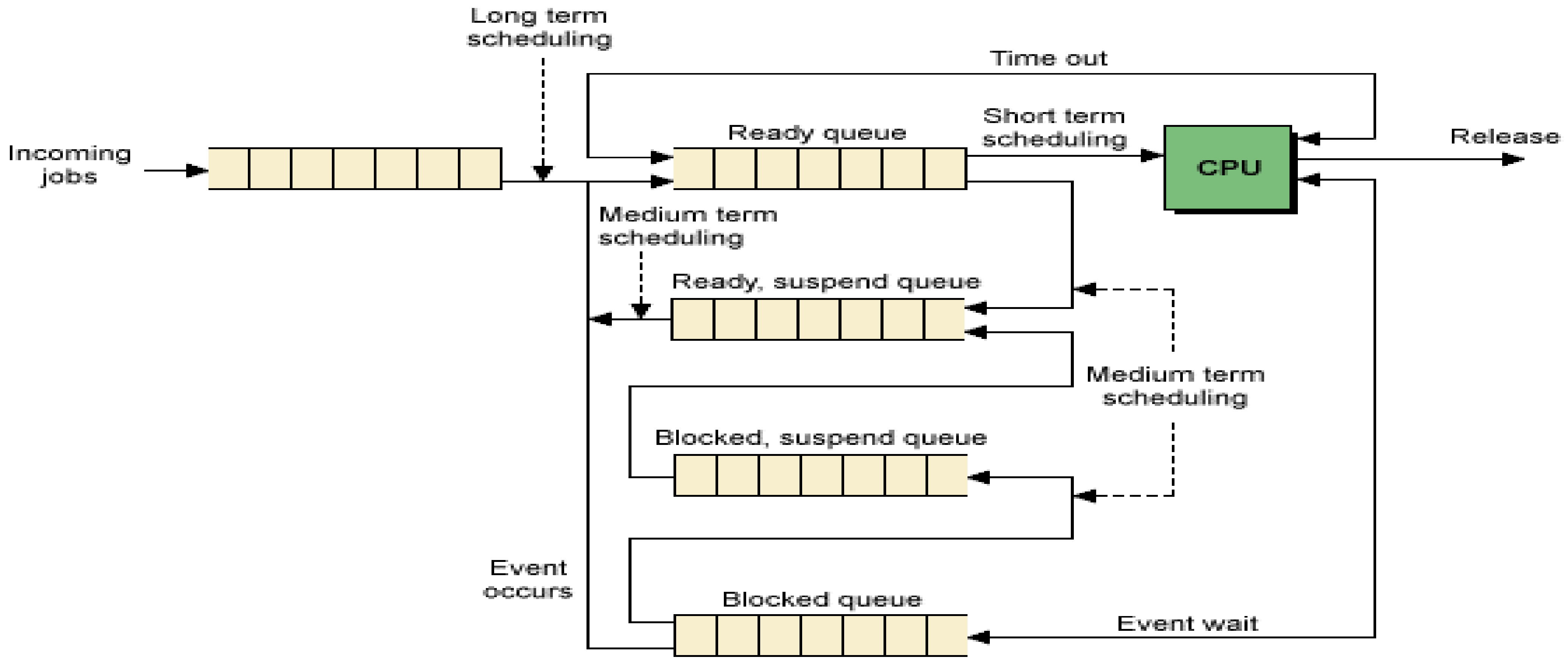
The Windows process management mechanisms:

- Windows creates processes in one-step by using Create Process. In Windows, there is no need to execute the process after its creation as it will already be executing the new code. However, the standard exec functions are still available in Windows.

There are three ways to carry out the process:

- 1- Use Hyper-V Manager on the host
- 2- Create a script in Windows PowerShell
- 3- Use Virtual Machine Manager (not included as part of Windows Server)

process management



process management topics

- Creating a New Process
- Replacing a Process Image (exec)
- Retrieving Process Information
- Waiting for a Spawned Process
- Processes vs. Threads
- Managing Process Resource Limits
- Limiting File I/O When Using Windows
- Process Accounting
- Managing and Scheduling Processes

Creating a New Process

the CreateProcess function enables the parent process to create an operating environment for a new process. The CreateProcess function creates a new process and its primary thread. For example by using a (spawn) function from the standard C runtime library we can port this code to Windows for a process using (CreateProcess) :

```
#include <Windows.h>
#include <process.h>
#include <stdio.h>
void main()
{
STARTUPINFO si;
PROCESS_INFORMATION pi;
GetStartupInfo(&si);
printf("Running Notepad with CreateProcess\n");
CreateProcess(NULL, "notepad", // Name of app to launch
NULL, // Default process security attributes
NULL, // Default thread security attributes
FALSE, // Don't inherit handles from the parent
0, // Normal priority
NULL, // Use the same environment as the parent
NULL, // Launch in the current directory
&si, // Startup Information
&pi); // Process information stored upon return
printf("Done.\n");
exit(0);
}
```

Thread Management

A thread is an independent path of execution in a process that shares the address space, code, and global data of the process. Time slices are allocated to each thread based on priority. Threads consist of an independent set of registers, stack, I/O handles, and message queue.

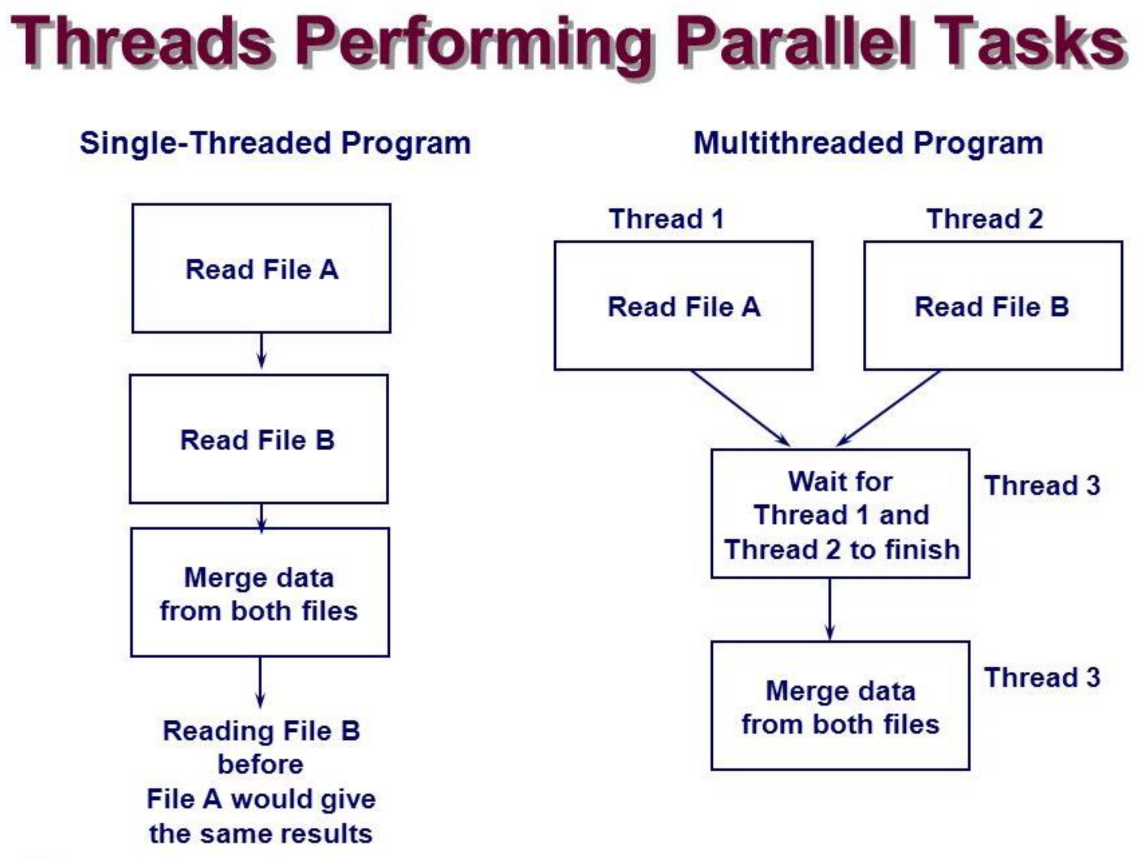
Threads can usually run on separate processors on multiprocessor computers. Windows enables you to assign threads to a specific processor on a multiprocessor hardware platform.

concept of threads

This section introduces the concept of threads. The following sections discuss the Windows APIs in managing threads:

- Creating a Thread
- Canceling a Thread
- Synchronization of Threads
- Thread Attributes
- Thread Scheduling and Prioritizing
- Managing Multiple Threads
- I/O Completion Ports

Thread diagram


Threads Performing Parallel Tasks
Single-Threaded Program
Read File A
Read File B
Merge data from both files
Reading File B before File A would give the same results
Multithreaded Program
Thread 1
Thread 2
Read File A
Read File B
Wait for Thread 1 and Thread 2 to finish
Thread 3
Merge data from both files
Thread 3

Creating a Thread

threads are created using the CreateThread function, which requires:

- The stack size of the thread.
- The security attributes of the thread.
- The address at which to begin execution of a procedure.
- A pointer to a variable to be passed to the thread.
- Flags that control the creation of the thread.
- An address to store the system-wide unique thread identifier.

After a thread is created, the thread identifier can be used to manage the thread (like get and set the priority of thread) until it has terminated. The next example demonstrates how you should use the **CreateThread** function to create a single thread.

Windows example: Creating a single thread

```
#include <Windows.h>
#include <stdio.h>
#include <stdlib.h>
char message[] = "Hello World";
DWORD WINAPI thread_function(LPVOID arg)
{
printf("thread_function started. Arg was %s\n",
(char *)arg);
Sleep(3000);
strcpy(message, "Bye!");
return 100;
}
void main()
{
HANDLE a_thread;
DWORD a_threadId;
DWORD thread_result;
// Create a new thread.
a_thread = CreateThread(NULL, 0, thread_function,
(LPVOID)message, 0,
&a_threadId);
if (a_thread == NULL)
{
perror("Thread creation failed");
exit(EXIT_FAILURE);
}
printf("Waiting for thread to finish...\n");
if (WaitForSingleObject(a_thread, INFINITE)
!= WAIT_OBJECT_0)
{
perror("Thread join failed");
exit(EXIT_FAILURE);
}
// Retrieve the code returned by the thread.
GetExitCodeThread(a_thread, &thread_result);
printf("Thread joined, it returned %d\n",
thread_result);
printf("Message is now %s\n", message);
exit(EXIT_SUCCESS);
}
```

summary

- Each process provides the resources needed to execute a program.
- A process has a virtual address space, executable code, open handles to system objects, a security context, a unique process identifier, environment variables, a priority class, minimum and maximum working set sizes, and at least one thread of execution.
- A thread is the entity within a process that can be scheduled for execution.
- All threads of a process share its virtual address space and system resources.
- Microsoft Windows supports preemptive multitasking, which creates the effect of simultaneous execution of multiple threads from multiple processes.
- A job object allows groups of processes to be managed as a unit. Job objects are namable, securable, sharable objects that control attributes of the processes associated with them.
- An application can use the thread pool to reduce the number of application threads and provide management of the worker threads.
- Applications can queue work items, associate work with waitable handles, automatically queue based on a timer, and bind with I/O.

References

- https://blogs.microsoft.com/microsoftsecure/2015/10/07/whats-new-with-microsoft-threat-modeling-tool-2016/
- https://technet.microsoft.com/en-us/library/bb497007.aspx
- www.buyya.com/microkernel/chap5.pdf
- https://www.d.umn.edu/~gshute/os/processes-and-threads.xhtml
- http://www.informit.com/articles/article.aspx?p=362660