# 

“Київський коледж зв’язку”

Циклова комісія Комп’ютерної інженерії

**Відповіді на контрольні запитання**

з дисципліни: «Операційні системи»

**Тема: “**ЕРУВАННЯ ПРОЦЕСАМИ ТА ПОТОКАМИ

(змістовий модуль). Процеси і потоки в сучасних ОС**”**

Виконав(ла) студент(ка)

групи РПЗ-83А

Хорошун Данило

Перевірив викладач

Повхліб В.С. \_\_\_\_\_\_\_

Київ 2020

Тема: КЕРУВАННЯ ПРОЦЕСАМИ ТА ПОТОКАМИ

(змістовий модуль). Процеси і потоки в сучасних ОС

# Лекція №4 (контрольні запитання)

Прочитати матеріал лекції та доповнити його відповідями на наступні питання (по 1 балу за кожне):

1. Класична модель потоків.

2. Потоки в POSIX

3. Реалізація потоків у просторі користувача

4. Реалізація потоків в ядрі

5. Гібридна реалізація потоків

Класична модель потоків

1) The classical model of flow

The process model is based on two independent concepts: the grouping of resources and their implementation. Sometimes it is useful to separate them, and here the streams come to the fore. A process is a way of grouping interdependent resources into a single whole. The process contains the address space of the program text and data, as well as other resources. These resources may include open files, raw alarms, signal handlers, accounting information, and more. Management of these resources can be significantly simplified by combining them as a process.

Another concept inherent in a process is the execution flow or simple flow. There is a command counter inside the stream, which monitors the execution of instructions. It contains registers with current operating variables. It has a runtime stack of one frame for each called procedure that does not return the control procedure. Although a flow can be performed in a process, the flow itself and its process are different concepts and should be considered separately. Processes are used to group resources into a single structure, and threads are the "value" allocated to the processor for execution.

Threads add to the process model the ability to perform multiple technological, often independent, tasks in a single process environment. Streams use the same address space, but different resources.

Like a traditional process (that is, a process with only one thread), the thread must be in one of the following states: executable, blocked, ready, or completed.

Note that each thread has its own stack, as shown in the figure. Each thread stack contains one frame for each procedure, which is called but not yet executed. This frame contains local variables and the address of the control that is returned when called. Typically, each thread requires different procedures, so each thread needs its own stack.

Some difficulties with working with threads:

1. If there are multiple threads in the parent process, should they be present in the subsidiary process?

2. If any of the parent process threads is blocked by a system call, for example, what happens to keyboard reading? Will there now be two streams that are forbidden to enter from the keyboard in the parent-child process? If a string is entered, will both streams be duplicated?

3. Another class of problems is the general data structure of flows. What happens if one thread closes the file and the other thread does not read data from it?

4. Suppose a thread notices a lack of free memory and begins to allocate additional volume. Halfway through, the threads shift, and the new thread notices that there is no free memory and begins to allocate additional volume. Additional memory can be allocated twice.

Потоки в POSIX

2) POSIX

POSIX or Pthread threads are a standard for entering POSIX threads that define an API for creating and managing threads.

Libraries that implement this standard and its functions are usually called Pthreads (functions are prefixed with "pthread\_"). Microsoft Windows (Pthreads-w32), which is widely used in Unix operating systems such as Linux or Solaris

Pthreads Defines a set of data types, functions, and constants in the C programming language format that are described in the pthread.h header file and implemented as a library.

All Pthreads procedures have names with the prefix "pthread\_" and can be divided into 4 categories depending on the purpose:

• flow management - creation, integration of flows, etc .;

• mutexic;

• conditional variables;

• Synchronize streams using locks and barriers to read / write data.

The POSIX API for semaphores works with POSIX threads, but is not necessarily part of the thread handling standard as defined in POSIX.1b, but Real-Time Extension). All semaphore procedures have the prefix "sem\_" instead of "pthread\_".

The main functions of the standard:

Data types:

pthread\_t: current descriptor

pthread\_attr\_t: set of thread attributes

Flow control functions:

pthread\_create (): create a thread

pthread\_exit (): end of thread

pthread\_cancel (): reject thread

pthread\_join (): block the flow until the end of another thread specified in the function call

pthread\_detach (): Release resources contained in the thread (if the thread is running, the release of resources will be after it is finished)

pthread\_attr\_init (): initialize the structure of the thread attributes

pthread\_attr\_setdetachstate (): An indication that the system can extract resources owned by the thread after the thread is completed

pthread\_attr\_destroy (): free memory from thread attribute structure (remove handle)

Stream synchronization features:

o pthread\_mutex\_init (), pthread\_mutex\_destroy (), pthread\_mutex\_lock (), pthread\_mutex\_trylock (), pthread\_mutex\_unlock (): use mutexes

o pthread\_cond\_init (), pthread\_cond\_signal, pthread\_cond\_wait (): use condy

Реалізація потоків у просторі користувача

3) Implementation of flows in the user space

If you place threads in user space, the kernel knows nothing about the presence of threads, which controls the usual single-thread processes.

The first and obvious advantage is that you can implement a set of user-level threads on an operating system that does not support threads.

Flows are executed at the top of the execution system, which is a set of procedures that control flows. We've looked at four of these, pthread\_create, pthread\_exit, pthread\_join, and pthread\_yield, but there are usually other procedures in the set.

When flow management is implemented in the user space, each process must have its own flow schedule that tracks the flows available in that process. This table is an analogue of the technological table available in the kernel, containing only the properties that belong to each chain.

The flow chart is managed by the program execution support system. When a thread is blocked or blocked, the information needed to restore execution is stored in the table, just as the kernel stores process information in the kernel process table. The replacement of threads performed in this way is at least sequential and possibly faster than interfering with kernel control, which is evidence of the benefits of a set of threads implemented at the user level.

The procedure that preserves the status of the thread and the scheduler are only local procedures, so calling them is much more efficient than the kernel. Among other things, there is no need to hold the kernel control with the trap command, no need to change the context, no need to restore the memory cache, and so on. Due to all this, the current scheduler works very fast.

Another advantage of implementing flows at the user level is the provision of each process planning algorithm with its own parameters.

Problems with streaming in the user space:

The complexity of implementing a call blocking system

Реалізація потоків в ядрі

4) Now let's look at the ability of the core to know and control flows. As shown in the figure, not everyone needs a runtime system. 2-8 (b). In addition, there is no technological schedule for each process. Instead, the kernel has a thread table that tracks all system flows. When a thread wants to create a new thread or delete an existing thread, it is called a kernel, and then creates or deletes the kernel by updating the flow chart.

The kernel sequence table contains registers for each state, state, and other information. Information is the same as user-level threads, but now it is in the kernel, not in the user's space (within the execution system). This information is a set of information that traditional cores store about each of their single-flow processes, ie the state of the process. In addition, the kernel supports the traditional process schedule for process monitoring.

All calls that can block the flow are treated as system calls, which is much higher than the system procedure during operation. When the thread is blocked, the kernel can, at its discretion, start another thread from that process (if it is ready) or a thread from another process. The user-level threads system will continue to run on its own until the kernel pulls the processor out of the processor (or there are no threads left to run).

Because it is relatively expensive to create and remove threads in the core, some systems resort to an environmentally friendly approach and recycle their threads. When a thread fails, it is marked as impossible to start, but its core structure is not affected otherwise. Later, when you need to create a new thread, the old thread is reconnected, which saves additional costs. Threading is also possible for user-level threads, but there is less incentive to do so because thread management costs are much lower.

Core streams do not require new, non-blocking system calls. In addition, if one thread in the process causes an error on the sheet, the kernel can easily check other running threads in the process and, if necessary, run one of them while waiting for the desired page to load. Their main disadvantage is that the cost of a system call is significant, so the more frequent the streaming operations (creation, completion, etc.), the higher the overhead costs.

Гібридна реалізація потоків

5)

Short-term research shows that the problem of efficiency in the PetaOps scale is widespread and easy to use thanks to innovative systems ten years later. The purpose of the study was to determine the feasibility and detailed structure of the parallel architecture, which combines the combined capabilities of semiconductor, superconductor and optical technology. This approach to hybrid architecture is based on the understanding that only a combination of technologies based on semiconductor technology can provide good performance attributes. An interdisciplinary team of staff was convened to develop the full structure of the system and implement the required point design to evaluate its performance using a small set of basic research programs.

The hybrid technology approach takes advantage of the important features offered by key new devices. In particular, with the latest advances in superconducting quantum logic with high-speed superconductivity, computing performance can be significantly improved, allowing 100 GHz for the next two years. Memory can be dramatically increased by a new hierarchy that combines high-density semiconductor memory and optical 3D holographic memory capable of storing 0.1 petabits or more in the future (albeit speculative). In optical networks with 100 Gbps channels, the interconnection capacity is significantly improved.

The simplicity of programming in the NUMA environment and the simplicity of designing the logic of the processor with a clock cycle of no more than 10 picoseconds is solved by quickly switching the context between the activation of memory threads using the multi-threaded architecture of the processor. This provides hardware delay management that can be adapted to a wide range of low-speed memory access. The physical memory hierarchy (possibly five levels) provides a balance between access demand and memory capacity. Using software control of the movement of blocks of variable parts in the memory structure, all elements of the structure will be fully visible in the system address space. Simple composite nuclear operations for interference will also be performed in memory.

The study uses applications with different characteristics, including a PPM code with a single static structure and a local connection, an adaptive FEM grid code with a time-varying non-uniform data structure, and a tree code for an N-gravitational body. non-uniform modeling with a PIC code that combines time-varying data structures and global connections, local and global connections with conventional static and unstable dynamic structures. The characteristics of these petaflop-like applications are taken from the analysis presented in previous workshops and used to assess computing needs for processor, memory, and communication resources.

Although general concepts, approaches and structures have been identified and analyzed, many details of architecture have not yet been identified. These are design parameters that define the dimensions of key components and low-level policies that manage resources to support parallel program execution. The main result of the study will be to determine the exact and effective values ​​of the design parameters and evaluate the results of work that can be achieved as a result of designing the system.