**Set 2**

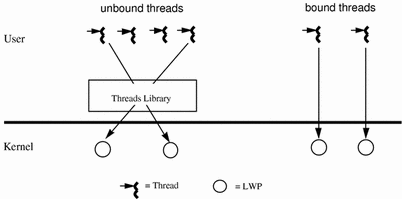
**Q3 Clearly differentiate b/w Process, Lightweight process, and Threads. What are process descriptors and different process states?**

Ans.

|  |  |  |  |
| --- | --- | --- | --- |
| **Comparison** | **Process**[[1]](#endnote-1) | **Threads**[[2]](#endnote-2) | **Lightweight Process (LWP)**[[3]](#endnote-3) |
| ***Definition/Need*** | A process is considered as an instance of a program in execution. | A thread is an entity within a process that can be scheduled for execution. | we need threads to share resources within a process, but we also need that kernel can pre-empt, schedule, and dispatch threads. Hence, a light-weight process (LWP) was introduced. |
| ***Implementation/Creation*** | Internally the clone () function is used to create a normal process.  That is to create a normal process fork () is used which further calls clone () with some relevant  arguments | pthread\_create () method can be used to create thread. To create a thread, a function from pthread  library calls clone () with appropriate flags. | Internally the clone () function is also used to create an LWP. To create an LWP, a function from pthread library calls clone () with appropriate flags. |
| ***Properties*** | When a process is created, it receives a copy of the parent’s address space and code execution start at the very next instruction following the process creation system call. Parent and child have  separate copies of the data (stack and heap) | All threads of a process share its system resources and virtual address space. Each thread maintains a unique thread identifier, a scheduling priority, exception handlers and thread local storage (TLB). | When an LWP is created, it shares all (or most of) its logical address space and system resources with other processes. |
| ***Use/Purpose*** | The purpose/use of a process is to behave as an entity to which some system resources like CPU time, memory, etc. are allocated. | Multiple threads are useful to exploit the parallelism possible on multiprocessor systems. | LWP bridge the user level and the kernel level. Linux uses lightweight processes to offer better support for multithreaded applications. |
| ***Communication*** | Process Communicate each other via Inter Process Communication. (IPC) | Since all the threads in the same program share the same memory segment of parent process thus, they access variables from data segment (Global variable) and thus communicate each other. | Synchronization of LWPs is maintained through mutex locks, condition variables, counting semaphores, and  reader-writer locks. A process cannot directly communicate with a specific LWP in another process |
| ***Performance/ Switching*** | Since switching from one normal process to another involves switching of all the process resources with those needed by a new process, so it is time consuming. All this  includes memory addresses, page tables, and kernel resources, caches in the processor. | Since thread switching involves switching out only identities and resources such as the program counter, registers and stack pointers, so it is very efficient and much cheaper. A thread can either be managed at the application level or kernel level. | During context switching LWPs does not involve the overhead of switching to the kernel address space.  An LWP is always managed by the kernel and is scheduled like a regular process. |

(References added in header column)

Fig. 2.1. User-level Threads and Lightweight Processes[[4]](#endnote-4)

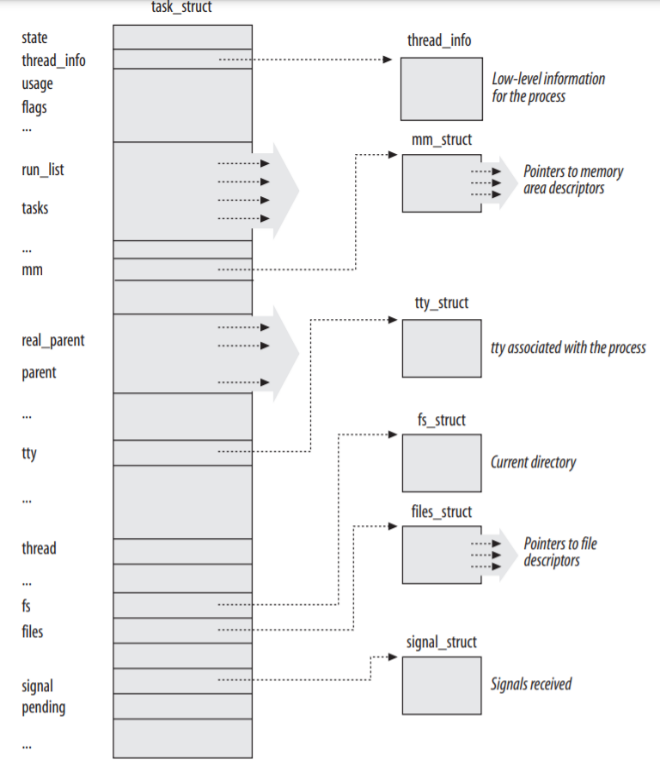


**Process Descriptor**

A process descriptor is a task\_struct type structure whose fields contain all the information related to a single process. These are stored in kernel address space and not in user space. The kernel stores the list of processes in a task list which is a circularly doubly linked list. Each element in the task list is called a process descriptor, and it has a struct of type task\_struct. It holds a lot of information about a process. Some of them are as follows:

|  |  |
| --- | --- |
| **FIELD IN TASK\_STRUCT DATA STRUCTURE** | **DESCRIPTION**[[5]](#endnote-5) |
| **state** | The state field of the process descriptor describes what is currently happening to the process. It consists of an array of flags, each of which describes a possible process state. |
| **ptrace** | It is primarily used to implement breakpoint debugging and system call tracing, that is, to trace system calls carried out by a process. |
| **static\_prio** | It is used for specifying process priority |
| **cpus\_allowed** | User can mention the CPU core for process execution |
| **mm** | mm field points to a mm\_struct structure which is the memory description (i.e., the physical memory pages and memory address space allocated to the process for a particular process.) |
| **exit\_code, exit\_signal** | It stores process exit information which helps the parent process to know how the child died |
| **pdeath\_signal** | The signal that is sent when the parent dies |
| **pid,uid,gid,environment** | These are process identifier, group id, user id and environment variables which are available to the process. |
| **parent, children** | It is a structure which points to parent process, and its children. |
| **utime,stime,cutime,**  **cstime** | It gives the information of user time, system time, collective user time, total system time spent by process and children |
| **files** | It is a structure which stores open file information. |
| **signals** | It handles some signals. For e.g., Pending signal |

Fig. 2.2 task\_struct[[6]](#endnote-6)



**Different Process states**

Below are the possible process states:

|  |  |
| --- | --- |
| **Different Process states** | **Description**[[7]](#endnote-7) |
| **TASK\_RUNNING** | The process is either executing on a CPU or waiting to be executed. |
| **TASK\_INTERRUPTIBLE** | Unless some condition become true, the process is in sleeping state or is suspended from being executed. |
| **TASK\_UNINTERRUPTIBLE** | It is similar to TASK\_INTERRUPTIBLE state but the sleeping process state remains same on sending a signal to it. This process state is rarely used only under some specific conditions in which a process must wait until a given event occurs without being interrupted. |
| **TASK\_STOPPED** | When the process execution is terminated. Generally, after receiving a kind of stop signal, the process enters in this state |
| **TASK\_TRACED** | Process enters in this state when the process execution has been stopped by a debugger. |
| **EXIT\_ZOMBIE** | When process execution is stopped without a wait4() or waitpid() system call, to return information about the dead process, being issued by parent process. |
| **EXIT\_DEAD** | Process enters this state when parent process has just issued a wait4() or waitpid() system call for removing the process by the system. It is final state. It is useful for avoiding race around conditions. |

Process states[[8]](#endnote-8)

Diagram

Description automatically generated

1. **Bovet, Daniel P., and Marco Cesati. Understanding the Linux Kernel: from I/O ports to process management. " O'Reilly Media, Inc.", 2005.** [↑](#endnote-ref-1)
2. [**https://docs.microsoft.com/en-us/windows/win32/procthread/about-processes-and-threads**](https://docs.microsoft.com/en-us/windows/win32/procthread/about-processes-and-threads) [↑](#endnote-ref-2)
3. [**https://www.andrew.cmu.edu/user/gkesden/ucsd/classes/sp16/cse120-a/applications/ln/lecture4.html**](https://www.andrew.cmu.edu/user/gkesden/ucsd/classes/sp16/cse120-a/applications/ln/lecture4.html) [↑](#endnote-ref-3)
4. [**https://docs.oracle.com/cd/E19455-01/806-5257/mtintro-72944/index.html**](https://docs.oracle.com/cd/E19455-01/806-5257/mtintro-72944/index.html) [↑](#endnote-ref-4)
5. <https://man7.org/linux/> [↑](#endnote-ref-5)
6. [**https://www.slashroot.in/difference-between-process-and-thread-linux**](https://www.slashroot.in/difference-between-process-and-thread-linux) [↑](#endnote-ref-6)
7. <http://www.linfo.org/process_state.html> [↑](#endnote-ref-7)
8. <https://subscription.packtpub.com/book/applicationdevelopment/9781785883057/1/ch01lvl1sec9/process-descriptors> [↑](#endnote-ref-8)