Universidad del Valle de Guatemala

Andrés Quan-Littow 17652

Lab#5

- Structs are what are called "User-defined data types" in C and C++, which allows to group items of many different types into a single type. To create a struct, one must use the following syntaxis:

```
struct myStructName
{
    char aCharVariable;
    char aCharArray[50];
    int  anIntVariable;
};
```

  They allow for the user to grab as much memory as needed inside a concise structure, allowing for the easy access, as it is a data type as well. They can be called in the same way an *int* data type, or other data types, can be called.

- Preprocessor directives are those that are included with the line #. These are invoked by the compiler to process some other programs before compilation. For example, we have the #include directive, and the #define/#undef directives.
- Pointers are literal pointers to memory. They are variables that save the direct address to somewhere else. They are usually used to pass references to arrays, among other objects, where a single value is not expected. Pointers have two kinds of operators: * and &. The first one is used to declare a pointer. For example, given $int * ip;$, we would expect the variable ip to point to an integer. Given any variable, and given any assignation, the & operand can be used to grab the address of any variable. On the other side, because they are complementary, * does the exact opposite. Given any address, if it is operated with a pointer *, then it will return the value stored at the given address.

- APT and dpkg both stand for special package-managing tools, they are especially popular in Debian-based distributions of Linux. APT stands for Advanced Package Tool, which allows for the installation and removal of core libraries and other software that are Debian-based. Dpkg, on the other hand, stands for "Debian PacKaGe". It allows the installation, building, removal, and management of Debian packages. It is also the replacement for aptitude in newer machines **(Kerrisk, 2021)**
- **Why did we modify the sched.h files?**
  The Sched.h headers define the sched_param structure, which contains the scheduling parameters required for implementation of each supported scheduling policy. By modifying them, we can change the scheduling policy of the machine. **(The Open Group, 1997)**
- **What is the purpose of the included definition, and of the other definitions in the file?**
  In specific, this is a headers file. A headers file allows a program to check for the existence of certain functions, variables, etc. Because a headers file is literally just a place to declare headers, it is also common to include other files in them, because header files often are on top of *.c programs, allowing them to be included first. Because they are included first, they allow for the inclusion of other files before the main program, allowing for the execution of such programs to be efficient and (mostly) error-free. An #define directive does nothing more but to define what can be understood as a global constant. In our case, the new constant SCHED_CASIO now has a value of 6, and, given the existence of CONFIG_SCHED_CASIO_POLICY, we define that same constant in the kernel's sched.h.
- **What is a task in Linux?**
  Tasks in the Linux Kernel are units of executions, of which may share various system resources with other tasks in the systems. They can be regarded as conventional threads or processes. **(The Open Group, 1997)**
- **What is the purpose of tast_struct and what is its equivalent on Windows?**
  The task_struct is a data structure with elements that can be described inside a circular doubly linked list, called the *task list*. **(InformIT, 2005).** On windows, these "processes" are represented in the *Executive Process Block* **(BoiseState, n.d.).**

- **What information does sched_param contain?**
  Sched_param describes scheduling parameters. It is used whenever the parameters for a thread or a process are needed (setters and getters) **(System Architecture, n.d.).**
- **What is the function rt_policy and what is the unlikely call in it?**
  The rt_policy is a function that tries to make it as unlikely as possible for the conditions described inside the *unlikely* functions to happen, based on the compiler. If they happen anyways, then it returns 0. **(INFORMATION, 2008)**
- **What kind of tasks are schedules by the EDF policy?**
  This kind of task processing stands for *Earliest Deadline First,* which attempts to schedule tasks based on their deadlines. **(Nyber & Hartman, 2016)**
- **Describe the priority precedence for EDF, RT and CFS policies**
  We must mention that EDF is *Earliest Deadline First,* RT is *Real-Time*, and CFS is *Completely Fair Scheduling.* Nevertheless, given what we have tried to do, it will try to make it so that FIFO goes first, SCHED_RR later, and finally SCHED_CASIO as last.
- **Explain the contents of casio_task.**
  Casio_task is a struct, so it is a distinct user data type that contains another two structs inside. It contains the node for it, and a casio_rb_node to distinguish it from other kinds of nodes. It also allows itself to have a deadline that can only be positive (hence the unsigned). It is of type long long. Then, it has another two structs, called list_head, for the head of the list, and the casio_list_node, that allows to know its own position in the list. Finally, it has a task struct pointer and a task. This pointer is to the task_struct, for easy positioning of the task. The task is what will be completed.
- **Explain the purpose and contents of the structure casio_rq.**
  Casio_rq seems to contain the rb_root and the casio_rb_root for the tasks and node management. Nevertheless, it also needs the head of the list, and the head of the list for the casio manager. It also has an atomic_t value to see if it is running. It appears to work as a checkpoint for the task to start running, allowing it to run only on certain intervals.
- **What is, and for what is atomic_t?**
  Atomic types allow the programmer to have an interface to the architecture's RMW operations between CPUs. It exists in two other shapes, and it allows for Arithmetic, bitwise

logic, swaps, reference counts, barriers, etc. **(Kernel, n.d.)**

- **What does the .next field entail in this structure?**
  The .next field points to the next value that is going to be needed, which is from the real time sched class. It will allow for the struct to "progress" through.
- **Why are the casio_tasks stored in a red-black tree and in a linked list?**
  Linked lists allow us to easily store dynamic-size arrays of data across a memory. This allows us to have as many tasks as we wish for. Nevertheless, the real worth of such a strategy is in the logic behind the rb tree. This tree, thanks to the linked list behavior, can be easily implemented. The rb tree allows us to move the earliest-deadline tasks to the left of the tree, allowing them to be chosen easily. Because it is a self-balancing binary tree, it helps the tasks to easily adapt to the demand of the program.
- **When does a casio_task preempt a task in execution?**
  It only stops it if the current's deadline is higher than the one that tries to call for the preempt.
- **What information does the system file contain that is used as an argument during the execution of casio_system?**
  The contents are:

50	duration						
1	2	2	16	16	16	5	5
2	3	3	15	15	15	5	5
3	2	2	14	14	14	5	5
4	3	3	15	15	15	5	5

  As we can see, it declares the duration of the whole simulation. It also marks 4 different jobs, as well as the different parameters (mins and max's) to be used. Deadlines are also present, as well as the seed. More in the files with the comments, on top of the main functions.
- **Explain how the scheduler SCHED_DEADLINE (introduced in Linux 3.14) adds an EDF algorithm to achieve temporal isolation.**
  The algorithm receives three parameters: runtime, period and deadline. Every time a task wakes up, the scheduler computes a scheduling deadline with a guaranteed, which are scheduled with an EDF strategy. This is because the runtime time units within deadline are a proper admission control strategy. This is the nature of EDF. **(Kernel)**

Works Cited

BoiseState. (n.d.). *Process Management*. Retrieved from Cs: http://cs.boisestate.edu/~amit/teaching/453/handouts/process-management-handout.pdf

INFORMATION, 1. (2008, September 20). *How do the likely/unlikely macros in the Linux kernel work and what is their benefit?* Retrieved from Stack Overflow: https://stackoverflow.com/questions/109710/how-do-the-likely-unlikely-macros-in-the-linux-kernel-work-and-what-is-their-ben

InformIT. (2005, January 31). *Process Management*. Retrieved from InformIT: https://www.informit.com/articles/article.aspx?p=368650#:~:text=The%20task_struct%20is%20a%20relatively,on%20a%2032%2Dbit%20machine.&text=The%20process%20descriptor%20contains%20the,more%20(see%20Figure%203.1).

Kernel. (n.d.). *Deadline Task Scheduling.* Retrieved from Kernel ORG: https://www.kernel.org/doc/Documentation/scheduler/sched-deadline.txt

Kernel. (n.d.). *On Atomic Types*. Retrieved from Kernel: https://www.kernel.org/doc/Documentation/atomic_t.txt

Kerrisk, M. (2021, 04 01). *dpkg*. Retrieved from Linux Manual Page: https://man7.org/linux/man-pages/man1/dpkg.1.html

Nyber, A., & Hartman, J. (2016). *Evaluation of EDF scheduling for Ericsson LTE system.* Linköping, Sweden: Linköping University.

System Architecture. (n.d.). *shed_param*. Retrieved from qnx: http://www.qnx.com/developers/docs/6.3.2/neutrino/lib_ref/s/sched_param.html

The Open Group. (1997). *Sched.h*. Retrieved from The Single UNIX Specification, Version 2: https://pubs.opengroup.org/onlinepubs/7908799/xsh/sched.h.html