**System programming**

**Two ways for Application to communicate with system/kernel:**

1. Application can make library call and then library call to system call
2. Application can directly make system call

Diagram

Description automatically generated

System call is controlled entry point into kernel code, allowing a process to request kernel to perform privileged operation. System call changes the CPU mode from user to kernel so that CPU can access protected memory. Each system call may have set of arguments that specify information to be transferred from user space to kernel space.

**How system call is handled:**

**Application makes system calls by invoking wrapper function of glibc(c library). It passes arguments by pushing them into stack in reverse order.**

**Wrapper function copies system call number to specific CPU registers. It also Copies arguments from process stack to CPU registers.**

**Now wrapper function executes the trap machine instruction, that causes CPU to switch from user mode to kernel mode. Trap handler saves system call no and arguments from CPU registers to process kernel stack, indexes system call table with system call no. to find address of system call routine. Trap handler transfers control to system call service routine.**

**System call executes and returns success or error. Trap handler again executes trap machine instruction to switch from kernel to user mode and returns control to wrapper function.**

Diagram

Description automatically generated

**Compilation and execution steps for c/c++ program:**

Diagram

Description automatically generated

**How to check functions available in library files:**

**ar -t libc.a | grep printf.o**

**Linking process: What linker do?**

**Relocation:** Merge code and data section of multiple object files into final object files.

**Symbol resolution:**

**Dynamic library linking can be done two times:**

**Load time (default case for Linux):** If library code is not already loaded by another process; loader loads the code into process’s address space. If library code is already loaded, then loader just puts the address

at place of unresolved reference inside memory.

**Run time:** Programmer has the responsibility to load theusing dlopen and dlclose**.** Executable is loaded into memory with unresolved references.

**Tools**:

**readelf - display information about ELF files**

**objdump - display information from object files**

**nm - list symbols from object files**

Diagram, timeline

Description automatically generated

**Static library:**

Whenever there is change in static lib, code must be recompiled as well but it is secured no other app can corrupt it. While dynamic lib is not part of executable but only one copy can be shared by multiple apps.

To create a dynamic library, write the following command:

gcc -g -fPIC -Wall -Werror -Wextra -pedantic \*.c -shared -o liball.so

The **-fPIC** flag allows the following code to be referenced at any virtual address at runtime. It stands for Position Independent Code.

export LD\_LIBRARY\_PATH=.:$LD\_LIBRARY\_PATH

**Static library creation:**

ar rc liball.a dog.o cat.o bird.o

ar is for archieving and -rc is for replace and create….

**Skipped video 4 and 5:**