VE482 Homework 1

Yihua Liu 518021910998 October 13,2021

Ex. 1 — Revisions

Explain the difference between the *stack* and the *heap*.

1. The allocation and de-allocation are manually done by programmers in help but automatically done by the compiler in stacks;
2. Accessing stacks is faster than accessing heap;
3. Heap frames are scattered in a large region of memory, while stack frame are concentrated on a small region of memory;
4. The heap is more flexible than stack because it can be resized;
5. Handling heap frames costs more than handling stack frames;
6. Stack is easy to overflow while heap is easy to cause fragmentation fault;
7. The implementation of a stack is easier than that of a heap;
8. Stack is thread safe because only the owner thread can access data, but heap is not thread safe because all threads can see data;
9. The data in stack frames are linearly stored while the data in heap frames are hierarchically stored. [1]

Ex. 2 — Personal research

1. Briefly explain what operations are performed when a computer is powered on. What is the role
   of the BIOS and how does it interact with the OS?
   When a computer is powered on, operations performed:
   After CPU powers on, it first finds BIOS in ROM or EPROM and runs it. BIOS or Boot Monitor does System startup. BIOS runs POST (Power On Self Test) in ROM, which initializes hardware devices by checking the BIOS, CMOS RAM, CPU, hardware devices, and secondary storage devices successively. After POST finishes, BIOS activates the computer's disk drives and finds the operating system. The bootstrap loaders (generally refers to bootloaders) load the operating system into memory and allows it to begin operation [2]. BIOS starts bootloaders, loading the kernel and initializing RAM disk. Master Boot Record (MBR) as Stage 1 bootloader can find the operating system; LILO, GRUB, etc. serves as Stage 2 bootloader. Finally, the operating system is initialized and daemons are started up [3].
   The role of BIOS is to run POST, initialize hardware devices, search for the MBR, find and initialize the operating system. It interacts with the OS by the bootloaders (generally refers to bootstrap loaders).
2. In a few words explain what are hybrid and exo kernels.
   Hybrid kernels has a similar structure to microkernel but are implemented in the manner of monolithic kernel. It is similar to monolithic kernel that operating system services are in the kernel space.
   Exokernels only allocates basic physical resources of the machine to application programs (library operating systems) that make use of the resources on their own, so that they can access hardware more directly.

Ex. 3 — Course application

1. Which of the following instructions should only be allowed in kernel mode? Explain.
   a) Disable all interrupts c) Set the time-of-day clock
   b) Read the time-of-day clock d) Change the memory map
   a) c) d).
   Interrupts are essential to the kernel. Users should not have the right to disable all interrupts. Users can read the time-of-day clock as information, but to set the time-of-day clock can cause application problems. To change the memory map may cause system to crash, so it should not be performed by users. Only the kernel can do the three operations in order to safely manage all the hardware resources.
2. Consider a system that has two CPUs and each CPU is composed of two threads. Suppose three programs, P0, P1, and P2, are started with run times of 5, 10 and 20 ms, respectively. How long will it take to complete the execution of these programs? Assume that all three programs are 100% CPU bound, do not block during execution, and do not change CPUs once assigned.
   If P0, P1, and P2 are executed by CPU 1 Thread 1, 2, and CPU 2 thread 1 respectively, the time is 20 ms.
   If P0, P2, and P1 are executed by CPU 1 Thread 1, 2, and CPU 2 thread 1 respectively, the time is 25 ms.
   If P1, P2, and P0 are executed by CPU 1 Thread 1, 2, and CPU 2 thread 1 respectively, the time is 30 ms.
   If P0, P1, and P2 are executed by CPU 1 Thread 1 and 2, the time is 35 ms.

Ex. 4 — Simple problem

One reason GUIs were initially slow to be adopted was the cost of the hardware needed to support them. How much video RAM is needed to support a 25 lines by 80 rows character monochrome text screen? How much for a 1024 x 768 pixel 24-bit color bitmap? Assuming the cost of this RAM in the 1980es was $5/KB what was the price of those two solutions? How much is it now?

First case:

$$(25 * 80)\text{bit} * (1/8)\text{Byte/bit} * (1/1024)\text{KB/Byte} = \frac{125}{512}\text{KB} \approx 0.244\text{KB}$$

$$\text{Price: } \frac{125}{512}\text{KB} * 5\$/\text{KB} = \frac{625}{512}\$ \approx 1.22\$$$

Second case:

$$(1024 * 768 * 24)\text{bit} * (1/8)\text{Byte/bit} * (1/1024)\text{KB/Byte} = 2304\text{KB}$$

$$\text{Price: } 2304\text{KB} * 5\$/\text{KB} = 11520\$$$

RAM is 7$/GB ~ 13$/GB now.

Take price is 10$/GB, first case price is almost ignorable, second case price is approximately 0.02$.

Ex. 5 — Command lines on a Unix system

```
#bin/bash

# 1. Create a new user
user add -m -g users yihua  # "users" can be replaced by "other"
# For version < 3.2.0, use
# adduser yihua other /home/yihua
# 2. List all the currently running processes
ps -x
# -a: Print all processes with controlling terminals
# -l: Give long listing
# -x: Include processes without a terminal
# -E: Print kernel endpoint numbers where pids are normally printed
# '-' is optional

# 3. Display the characteristics of the CPU and the available memory
cat /proc/cpuinfo
cat /proc/meminfo
# For dynamic display, try `top` command

# 4. Redirect some random output into two different files
head -n 1 /dev/urandom | tee 1.txt > 2.txt
# Use /dev/urandom rather than /dev/random ('u' means 'unlimited'), see
# https://unix.stackexchange.com/questions/324209/when-to-use-dev-random-vs-dev-urandom
# https://www.2uo.de/myths-about-urandom/
# The original stream is 2-base
# Adding an `od` pipe before `head` after `tee` can directly convert to other-base digits

# 5. Concatenate the two previous files
cat 1.txt 2.txt > 3.txt

# 6. Read the content of the resulting file as hexdecimal values (in other words find a command to read a file as hexadecimal values)
od -x 3.txt
# Usage: od [-bcdhovx] [file] [ [+] offset [.] [b] ]
# -x: Dump words in hex
# or `hexdump 3.txt`

# 7. Use a single command to find all the files in `/usr/src` with the word `semaphore` in their name and containing the word `ddekit_sem_down`
find -X /usr/src -name "*semaphore*" | xargs grep -lw 'ddekit_sem_down'
# Different from Linux, `grep` of Minix3 cannot accept stdin as arguments, so `xargs` is needed
# Double quotes are recommended for `find` while single quotes are recommended for `grep`
# The `-X` option is a modification to permit `find` to be safely used in conjuction with xargs(1). If a file name contains any of the delimiting characters used by `xargs`, a diagnostic message is displayed on standard error, and the file is skipped.
# Alternatively, the `-print0` or `-printx` primaries can be used to format the output in a way that `xargs` can accept.
```

1. Ankit_Bisht. "Stack vs Heap Memory Allocation". GeeksforGeeks. Jun. 21, 2021. URL: https://www.geeksforgeeks.org/stack-vs-heap-memory-allocation/. ↩

2. Curt Franklin and Dave Coustan. "How Operating Systems Work: Computer Operating Systems". HowStuffWorks. Aug. 14, 2000. URL: https://computer.howstuffworks.com/operating-system4.htm. ↩

3. Saket Kumar. "What happens when we turn on computer?". GeeksforGeeks. Mar. 25, 2021. URL: https://www.geeksforgeeks.org/what-happens-when-we-turn-on-computer/. ↩