**Assignment – 1**

**Question 1 (20%): Comparing the Cost of Output with Three I/O Models:**

**The textbook discussed three I/O modes: Programmed I/O, Interrupt-driven I/O, and DMA. Answer the following questions:**

1. **What are the main differences between the three I/O models and what are their strength and weakness?**

### Programmed I/O

Programmed I/O is the most simple technique for exchanging data or communication b/w the CPU and any other external device. Data is sent and received by the processor and the I/O module. The processor gives complete control to the I/O operation i.e. sensing device status, sending READ/WRITE command. Once the processor issues a command to the I/O module, it then waits for the operation to execute and complete. However, the processor confirms the status of the operation till the I/O module is executed completely.

**Strengths:**

* Programmed I/O is simple to implement.
* Requires very little hardware support.

**Weakness:**

* Longer wait time.
* Ties up the CPU for longer time with no

### Interrupt Driven:

Interrupt I/O is a way of controlling I/O jobs where the external device/peripheral/ command line sends a signal about the due work. Hence, it causes an interrupt to be set. So, in interrupts have a priority level in the processor. Therefore, it requires more complex system and software, however, is a lot more efficient than programmed I/O in terms of efficiently using computer’s time and processing capacity.

**Strength:**

* It is fast because the I/O devices only interrupt when in ready state.
* It is efficient because it lets the CPU perform other tasks while it is waiting for an interrupt.

**Weakness:**

* CPU spends most of the time in a loop, waiting fir the device to become ready, called *busy waiting*.
* Requires an interrupt for every character read/write. Interrupting a running process is can be expensive operation.

### Direct Memory Access (DMA)

DMA is a technique of transferring data within main memory and external device w/o passing it through the CPU. DMA improves the processing capacity of the operating system because it takes automates the task of transferring data from the processor and let it perform other tasks.

**Strength:**

* Allows the peripheral devices to READ/WRITE without going through the CPU.
* Frees the processor from transferring data, hence, allows faster processing since, the processor can be doing something else.

**Weakness:**

* When peripheral device wants to READ/WRITE it makes the CPU wait. This is called the *cycle stealing*.

### Key Differences:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Programmed I/O** | **Interrupt-driven I/O** | **DMA** |
| **1.)** |  |  | Direct Memory Access (DMA) is a system in which a hardware component |
| **2.)** |  |  |  |
| **3.)** |  |  |  |

1. **Assuming that an application needs to output 1000 words from the internal memory to the hard disk, calculate the following values for each I/O model:**
   * **How many times the processor is interrupted?**
   * **How many times the internal memory is read by the processor for those 1000 words?**
   * **How many times the disk controller is read by the processor?**
   * **How many times the disk controller is written to by the processor?**

**Draw a table to contain your answers.**

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Justification:

**2. (10 marks) Based on the information generated from the above program, produce a memory map table showing the layout of literals, initialised global variables, uninitialized global variables, formal parameters of each function, local variables, dynamically allocated variables, functions, environment and command line arguments in the memory when the program runs at label L in function f2.**

**The memory map table must show the addresses of each variable, literal, and function and their sizes. It should also show the start**

**and end addresses of the environment and the command line arguments and their sizes.**

**The memory map table must contain at least the following columns:**

**1. The start address of an entity such as a variable or a function**

**2. The length of the storage space of the entity in bytes**

**3. The name of the entity, such as global\_pointer1 or Hello, world!**

**4. The nature of the entity, such as *function*, or *uninitialized global variable***

**5. The memory section, e.g. environment, command line arguments, code (or text), global initialised data, global uninitialed data, stack, heap etc.**

**In addition, you must use seven different background colours to highlight the following seven memory sections as indicated below::**

**[Green] initialised global variables (including constants and literals)**

**[Red] uninitialised global variables**

**[Blue] stack (containing the local variables and returning addresses of function calls)**

**[Magenta] heap (containing the dynamically allocated memories)**

**[Yellow] code (functions)**

**[Cyan] process environment**

**[White] command line arguments**

**Please also note that in your memory table, *the memory addresses must be strictly sequential, from the highest address to the lowest address* to reflect how different components of a running program are layout in the virtual memory. Your memory tablewill not be accepted if the addresses are not lined up sequentially in the table. If you find that components from one section aresplit in more than one continuous area of memory, it is a sure indication that there is something wrong with your memory map andyou should find out what went wrong and fix it.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **entity name such as global\_pointer1 or Hello, world!** | **Start address** | **Length of Storage (byte)** | **Nature of entity (Memory type) *such as function*, or *uninitialized global variable*** | **Memory section e.g. environment, command line arguments, code (or text), global initialised data, global uninitialed data, stack, heap etc** |
| argc | 0x7fffc604dffc | 4 | Length of command line argument | Command line argument |
| Argv | 0x7fffc604e0f8 | 8 | Command line arguments | Command line argument |
| &environ[0] | 0x7fffc604e138 |  | Process environment | Environment |
| &environ[sizeof  (environ-1)] | 0x7fffc604e170 |  | Process environment | Environment |
| &main | 0x7f1b790009b6 |  | Function | Initialised data |
| &f1 | 0x7f1b790008c9 |  | Function | Initialised data |
| &f2 | 0x7f1b7900081a |  | Function | Initialised data |
| global\_x | 0x7f1b79202010 | 4 | Global Initialised variable |  |
| global\_y | 0x7f1b79202050 |  | Un-initialised global variable |  |
| global\_array1 | 0x7f1b79202018 |  | initialised global array |  |
| global\_array2 | 0x7f1b79202058 | 10 | Un-initialised global array |  |
| F1\_x | 0x7fffc904ec40 | 4 |  | Stack |
| f1\_y | 0x7fffc904ec44 | 4 | Un-initialised variable | Stack |
| f1\_p1 | 0x7fffc904ec48 | 8 |  | Stack |
| f1\_p2 | 0x7fffc904ec50 | 800 | Local dynamic array | Heap |
| X | 0x7fffc904ebec | 4 |  |  |
| x1 | 0x7fffc904ec3c | 4 |  | Stack |
| x2 | 0x7fffc904ec38 | 4 |  | Stack |
| x3 | 0x7fffc904ec34 | 4 |  | Stack |
| x4 | 0x7fffc904ec30 | 1 |  | Stack |
| x5 | 0x7fffc904ec28 | 8 |  | Stack |
| x6 | 0x7fffc904ec24 | 4 |  | Stack |
| f2\_p | 0x7fffc904ec00 | 400 | Dynamically allocated memory | Heap |
| f2\_x | 0x7fffc904ebfc | 4 | Local integer | Stack |
| f2\_p[0] | 0x7fffc06974e0 | 1 | First character of the f2\_p | Heap |

**Output listing:**

root@vishnu-rana:/mnt/c/Users/Vishnu Rana/Documents/ICT374-OSSP/Assignment/q2# gcc memory.c -o memory

root@vishnu-rana:/mnt/c/Users/Vishnu Rana/Documents/ICT374-OSSP/Assignment/q2# ./memory /bin/ls /bin/ps /bin/date /bin/who /bin/uname

My OS bit size: 64

Address of argc 0x7fffc904ec7c

Address of all elements in argv

address from argv element 0 = 0x7fffc904ed78

address from argv element 1 = 0x7fffc904ed80

address from argv element 2 = 0x7fffc904ed88

address from argv element 3 = 0x7fffc904ed90

address from argv element 4 = 0x7fffc904ed98

address from argv element 5 = 0x7fffc904eda0

Address of the first command line arguments 0x7fffc904ed78

Address of the last command line arguments 0x7fffc904eda0

First address of environment variable 0x7fffc904edb0

Last address of environment variable 0x7fffc904ede8

Starting address of function main = 0x7f5d788009b6

Starting address of function f1 = 0x7f5d788008c9

Starting address of function f2 = 0x7f5d7880081a

address of global\_x = 0x7f5d78a02010

address of global\_y = 0x7f5d78a02050

address of global\_array1 = 0x7f5d78a02018

address of global\_array2 = 0x7f5d78a02058

address of global\_pointer1 = 0x7f5d78a02030

address of global\_pointer2 = 0x7f5d78a02048

address of global\_float = 0x7f5d78a02028

address of global\_double = 0x7f5d78a02068

Address of string literal hello = 0x7f5d78a02018

Address of string literal bye = 0x7f5d78a02030

location x1 = 0x7fffc904ec3c

location x2 = 0x7fffc904ec38

location x3 = 0x7fffc904ec34

location x4 = 0x7fffc904ec30

location x5 = 0x7fffc904ec28

location x6 = 0x7fffc904ec24

f1\_x address = 0x7fffc904ec40

f1\_y address 0x7fffc904ec44

f1\_p1 address = 0x7fffc904ec48

f1\_p2 address = 0x7fffc904ec50

Address of string literal (this is a string) 0x7fffc904ec48

x location = 0x7fffc904ebec

f2\_p location = 0x7fffc904ec00

f2\_x location = 0x7fffc904ebfc

f2\_p location = 0x7fffc06974e0

root@vishnu-rana:/mnt/c/Users/Vishnu Rana/Documents/ICT374-OSSP/Assignment/q2#

**(5 marks) Based on the experiment and analysis you have carried out in 1 and 2 above, answer the following questions:**

1. **what is the size of the virtual address space of that process?**
2. **how does the operating system on your machine layouts the following process components in the virtual address space: command line arguments, environment, literals, initialised global variables, uninitialised global variables, functions, formal parameters and local variables of a function, and dynamically allocated variables (or memories)?**

My personal computer use segmentation instead of paging, dividing virtual address spaces into variable-length segments. Hence, a address consists of a segment number and an offset within the segment.

### Question 3 (25%): Executing Commands in Child Processes

Write a program that takes a list of command line arguments, each of which is the full path of a command (such as /bin/ls, /bin/ps, /bin/date, /bin/who, /bin/uname etc). Assume the number of such commands is N, your program would then create N direct child processes (ie, the parent of these child processes is the same original process), each of which executing one of the N commands. You should make sure that these N commands are executed concurrently, not sequentially one after the other. The parent process should be waiting for each child process to terminate. When a child process terminates, the parent process should print one line on the standard output stating that the relevant command has completed successfully or not successfully (such as "Command /bin/who has completed successfully", or "Command /bin/who has not completed successfully"). Once all of its child processes have terminated, the parent process should print "All done, bye-bye!" before it itself terminates.

**Question 4 (25%): Reporting Information of Files**

**Write a C program, myls.c, that is similar to the standard Unix utility ls -l (but with much less functionality). Specifically, it takes a list of command line arguments, treating each command line argument as a file name. It then reports the following information for each file:**

1. **user name of the owner owner (*hints: Stevens & Rago, 6.2.*);**
2. **group name of the group owner; (*hints: Stevens & Rago, 6.4.*);**
3. **the type of file;**
4. **full access permissions, reported in the format used by the ls program;**
5. **the size of the file;**
6. **I-node number;**
7. **the device number of the device in which the file is stored, including both major number and minor number (*hints: Stevens & Rago, 4.23.*);**
8. **the number of links;**
9. **last access time, converted to the format used by the ls program (*hint: Stevens & Rago, 6.10.*);**
10. **last modification time, converted to the format used by the ls program;**
11. **last time file status changed, converted to the format used by the ls program;**

**Like ls -l command, if no command line argument is provided, the program simply reports the information about the current directory.**

**Of course, your program cannot use ls program in any way.**