Project - 2

Design Document

Vignesh Kulothungan ([vxk111430@utdallas.edu](mailto:vxk111430@utdallas.edu))

Date: 25th February, 2014

**Algorithm:**

**Parent Process:**

* Registers an interrupt handler for SIGUSR1 signal.
* Reads **port numbers (all three port numbers), main process hostname (IP) and Delay** values from **‘config.dat’**.
* First port number belongs to parent and other two belongs to child processes.
* Creates an UDP socket to receive acknowledgments from child processes. Parent is the server for this UDP socket.
* Then waits for child processes to create their sockets and tries to connect to both ‘adder’ and ‘factorial’ TCP sockets. Parent is the client for these TCP sockets.
* After connecting, parent waits for synchronization messages from both child processes.
* After receiving the sync messages, parent starts reading from ‘instruction.dat’ and sends it to corresponding child process.
* Before sending each instruction to the child, parent checks if the interrupt flag is set. The interrupt flag is set if the parent had received SIGUSR1 from any of the child processes.
* If the interrupt flag is set, then parent blocks in the acknowledgement UDP socket, waiting for ‘start’ message from the child which sent the SIGUSR1 signal.
* Until the child process which sent the signal acknowledges, the parent will be blocked.
* After sending the instruction to the child, parent sends a SIGUSR1 signal to the child indicating that an instruction has been sent.
* A delay is induced after sending the signal, to allow the child’s interrupt handler to execute. This is done because similar signals received are not queued by the kernel, if a handler is executing and if the same signal is received, then the signal is discarded. To avoid this, a delay is induced.
* After reaching the end of the ‘instruction.dat’ file, the parent exits the loop and waits for the child processes to exit.
* Once the child processes exit, parent terminates.

**Parent Process Interrupt Handler:**

* Sets the interrupt flag to 1. The parent in main routine will check this flag before sending each instruction.
* Increments the interrupt count. This counter makes sure that parent blocks twice if both factorial and adder had sent SIGUSR1 requesting the parent to stop sending instructions. In this case, parent waits for two messages (one from each process) in the main routine.

**Adder Process:**

* Registers an interrupt handler for SIGUSR1 signal.
* Reads **parents port number, adder hostname, adder port number, N and M** from ‘config.dat’ file.
* Adder creates a TCP socket to receive instructions from the parent. It waits for the parent to connect and then accepts the connection. After a connection is made, it sends a synchronization message to the parent. Adder is the server for this TCP socket.
* Adder then connects to the UDP socket created by parent, to send acknowledgment messages. Adder is the client for this UDP socket.
* Enters loop to receive instructions from parent.
* If received filename is “stopstop”, then adder exits.
* Else opens the file. If file is not found, then skips the current instruction and waits for the next one.
* Else reads the first line to get the number of integers to be added.
* Then reads all the integers from file in a loop and adds it to a sum variable.
* Once all integers are read and added to the sum, it exits the loop.
* Sum is a ‘long long int’ type variable, it is used to handle integer overflows. (‘int’ is 4 bytes and ‘long long’ is ‘8 bytes’).
* The output is displayed, increments the counter which keeps track of number of completed instructions. It checks if lower instruction limit ‘M’ is reached. If (received – completed <= M) and if the parent was signaled to stop sending instructions earlier, adder sends a ‘start again’ acknowledgement message to make the parent start sending instructions again.
* Then waits in queue for next instruction.

**Adder Process Interrupt Handler:**

* If SIGUSR1 signal was sent by parent, then adder’s interrupt handler executes. It increments the number of instructions received counter and checks if (received – completed >= N) then a SIGUSR1 signal is sent to parent to stop it from sending more instructions.

**Factorial Process:**

* Registers an interrupt handler for SIGUSR1 signal.
* Reads **parents port number, facto hostname, facto port number, N and M** from ‘config.dat’ file.
* Factorial process creates a TCP socket to receive instructions from the parent. It waits for the parent to connect and then accepts the connection. After a connection is made, it sends a synchronization message to the parent. Factorial is the Server for this TCP socket.
* It then connects to the UDP socket created by parent, to send acknowledgment messages. It is a client for this UDP socket.
* Enters loop to receive instructions from parent.
* Reads data from parent in a while loop.
* If received integer values are zeroes, then factorial exits.
* Else, calls facmod() function to calculate the factorial of number1 and (factorial(number1) mod number2). Then, prints the output.
* It increments the counter which keeps track of number of completed instructions. It checks if lower instruction limit ‘M’ is reached. If (received – completed <= M) and if the parent was signaled to stop sending instructions earlier, factorial sends a ‘start again’ acknowledgement message to make the parent start sending instructions again.
* Then waits in queue for next instruction.

**Factorial Process Interrupt Handler:**

* If SIGUSR1 signal was sent by parent, then factorial’s interrupt handler executes. It increments the number of instructions received counter and checks if (received – completed >= N) then a SIGUSR1 signal is sent to parent to stop it from sending more instructions.

**TCP Socket configuration:**

TCP Socket created by Adder

Server

Client

Factorial

Adder

Parent

Server

TCP Socket created by Factorial

**UDP Socket configuration:**

Client

Server

Adder

Parent

Client

UDP Socket created by Parent

Factorial

**Signal Flow:**

Adder

Parent

SIGUSR1, if limit N is reached

SIGUSR1, Sent after sending an Instruction.

Factorial

SIGUSR1, if limit N is reached

**Error handling limitations:**

* Filename in add instruction cannot be more than 8 bytes.
* Integer values cannot be more than 2^30.
* No more factorial instructions in instruction.dat, once “fac 0 0” is sent.
* No more addition instructions in instruction.dat, once “add stopstop 0” is sent.
* Parent will not know what data is sent to child, it just checks if the operation is fac or add. It does not know if “stopstop” or “filename” is sent.
* Hostname is ‘config.dat’ is ‘localhost’ for all three processes, since they all run in the same system.

**Data structures used:**

* facmod() function is used to calculate factorial and modulus.