**Lab1 Report**

1. **Results on Linux**

Initialization Time:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | Linux Initialization Time | | | | |
| B | | | | |
| 1 | 2 | 4 | 8 | 10 |
| N | 20 | 0.000538 | 0.000525 | 0.000528 | 0.000533 | 0.000534 |
| 40 | 0.000525 | 0.000537 | 0.000547 | 0.000538 | 0.000523 |
| 80 | 0.000553 | 0.000518 | 0.000554 | 0.000505 | 0.000551 |
| 160 | 0.000514 | 0.000556 | 0.000504 | 0.000553 | 0.000535 |
| 320 | 0.000517 | 0.000550 | 0.000529 | 0.000516 | 0.000546 |

Initialization time does not vary with the increase of message size and number of messages, and it is quite consistent.

Data Transmit Time:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | Linux Data Transmission Time | | | | |
| B | | | | |
| 1 | 2 | 4 | 8 | 10 |
| N | 20 | 0.001704 | 0.001718 | 0.001687 | 0.001707 | 0.001709 |
| 40 | 0.002996 | 0.003155 | 0.003195 | 0.003145 | 0.003095 |
| 80 | 0.005066 | 0.00564 | 0.004478 | 0.005037 | 0.005307 |
| 160 | 0.005316 | 0.003983 | 0.004346 | 0.006741 | 0.005929 |
| 320 | 0.0105 | 0.009319 | 0.00695 | 0.005223 | 0.009177 |

Data transmission time increases as the number of messages increases. However, the time does not vary with the increase of message queue size.

Standard Deviation of the Data Transmission Time:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | Standard Deviation of the Data Transmit Time | | | | |
| B | | | | |
| 1 | 2 | 4 | 8 | 10 |
| N | 20 | 0.00001543 | 0.00002454 | 0.0007547 | 0.00001388 | 0.0000313 |
| 40 | 0.0001165 | 0.00009349 | 0.00008796 | 0.00004275 | 0.0001078 |
| 80 | 0.000779 | 0.0002639 | 0.0006402 | 0.0005206 | 0.0001241 |
| 160 | 0.0003738 | 0.0003946 | 0.0009075 | 0.0009887 | 0.002657 |
| 320 | 0.000329 | 0.001744 | 0.001015 | 0.0004651 | 0.0003431 |

The following histogram shows the distribution of the average data transmission time given that N = 320 and B = 10. Data is collected by running the program 50 times, and most of the measured transmission time fall into the range of 0.008 – 0.009 and 0.010 – 0.020.

1. **Results on Keil LPC1768 Board**

Initialization Time:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | Keil Initialization time | | | | |
| B | | | | |
| 1 | 2 | 4 | 8 | 10 |
| N | 20 | 0.00001 | 0.00001 | 0.00001 | 0.00001 | 0.00001 |
| 40 | 0.00001 | 0.00001 | 0.00001 | 0.00001 | 0.00001 |
| 80 | 0.00001 | 0.00001 | 0.00001 | 0.00001 | 0.00001 |
| 160 | 0.00001 | 0.00001 | 0.00001 | 0.00001 | 0.00001 |
| 320 | 0.00001 | 0.00001 | 0.00001 | 0.00001 | 0.00001 |

Initialization time does not vary with the increase of message size and number of messages, and it is consistent.

Data Transmission Time:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | Keil Data Transmission time  (Timer Tick Value = 10us) | | | | |
| B | | | | |
| 1 | 2 | 4 | 8 | 10 |
| N | 20 | 0.00001 | 0.00001 | 0.00001 | 0.00001 | 0.00001 |
| 40 | 0.00001 | 0.00001 | 0.00001 | 0.00001 | 0.00001 |
| 80 | 0.00002 | 0.00002 | 0.00001 | 0.00001 | 0.00001 |
| 160 | 0.00004 | 0.00003 | 0.00002 | 0.00001 | 0.00001 |
| 320 | 0.00005 | 0.00003 | 0.00002 | 0.00002 | 0.00002 |

Data transmission time increases as the number of messages increases and decreases as the mailbox size increases.

Since our program is executed in a very short amount of time, it is impossible to measure the system initialization time and data transmission time with the timer tick value set to the default value, which is 10000us. Instead of using the default timer tick value, we set to timer tick value to 10us, so that the system initialization time and data transmission time can be measured. Our program measures the system tick value before the send task is initiated, before the first integer is generated, and after the last integer is consumed. And by looking at the system tick value difference, we can calculate the approximate system initialization time and data transmission time by using *tick value difference/100000*. Since the program is unable to display any messages on the GLCD monitor after changing the timer tick value to 10us, the value shown on the above table is collected by using the debugger’s watch window.

By observation, the system initialization time and data transmission time are quite consistent. Therefore, it is not necessary to execute the program multiple times for a given (N, B) pair in order to compute the average transmission time.

1. **Comparison of Test Results on Linux and LPC1768**

By comparing the Linux initialization time table in part 1 with the Keil LPC1768 board initialization time table in part 2 of this report, it is obvious that the time needed for initialization on the Linux platform is longer that the time needed on the LPC1768 board. Similar to the initialization time measured on the Linux platform, the Keil LPC1768 board‘s initialization time does not vary with the increase of mailbox size. However, the program that executes on the Linux platform clearly needs more time to initialization that the program executes on the LPC1768 board.

Similarly, by comparing the Linux data transmission time table in part 1 with the Keil LPC1768 board data transmission time table in part 2, it is clear that the data transmission time on the Linux platform is longer than the data transmission time on the Keil LPC1768 board. The data transmission time measured on the Keil LPC1768 board increases linearly with the increase of the number of messages and decreases linearly with the increase of the mailbox size without noticeable time deviation. Although the data transmission time measured on the Linux platform does increase almost linearly with the increase of the number of messages, the data transmission time does not decrease as the message queue increases, and there is noticeable data transmission time deviation that performance is not consistent.

To be concluded, the Keil LPC1768 board is better than the Linux platform in terms of inter-process communication through message passing since the time needed for initialization and data transmission on the Keil LPC1768 board is clearly less that the time needed on the Linux platform. However, the Keil LPC1768 board is definitely more costly than the Linux platform since it is so much faster in message passing than the Linux platform.

**Appendix**

**1. Program on Keil LPC1768 Board**

#include <LPC17xx.h>

#include <RTL.h>

#include "GLCD.h"

#include <stdio.h>

int n = 10; // Number of messages.

int b = 1; // Mailbox size.

int tick; // System tick.

OS\_TID id1; // task ID for event transmit.

os\_mbx\_declare (MsgBox, 1); // Declare an RTX mailbox with b entries

U32 mpool[1\*(2\*sizeof(U32))/4]; // Reserve a memory for b messages

\_\_task void rec\_task (void);

\_\_task void send\_task (void) {

/\* This task will send a message. \*/

U32 \*mptr;

char buffer[20];

int i = 0;

os\_tsk\_create (rec\_task, 1); // Create and start receiver.

os\_mbx\_init (MsgBox, sizeof(MsgBox)); // Initialize mailbox.

// Get system time tick.

tick = os\_time\_get() - tick;

sprintf(buffer, "Init system: %d ticks", tick);

GLCD\_DisplayString(0,0,1,buffer);

while(i < n) {

mptr = \_alloc\_box (mpool); //Allocate a memory for the message

mptr[0] = rand()%50; //Set the message content.

mptr[1] = 0x0;

// Send message.

if (os\_mbx\_send (MsgBox, mptr, 0x1) == OS\_R\_TMO) {

\_free\_box (mpool, mptr);

} else {

i++;

}

}

os\_evt\_wait\_or (1, 0x000f); // Wait for receiver to terminate.

os\_tsk\_delete\_self ();

}

\_\_task void rec\_task (void) {

/\* This task will receive a message. \*/

U32 \*rptr, rec\_val[2];

int rec;

char buffer2[20];

int nCount = 0;

// Receive Messages.

while(nCount < n) {

rec = os\_mbx\_wait (MsgBox, (void\*\*)&rptr, 0xf);

if (rec == OS\_R\_OK || rec == OS\_R\_MBX) {

rec\_val[0] = rptr[0];

rec\_val[1] = rptr[1];

\_free\_box (mpool, rptr);

nCount++;

}

}

// Get system time tick.

tick = os\_time\_get() - tick;

sprintf(buffer2, "Data transfer: %d ticks", tick);

GLCD\_DisplayString(0,0,1,buffer2);

// Notify sender that receiver is going to terminate.

os\_evt\_set (1, id1);

os\_tsk\_delete\_self ();

}

void main (void) {

SystemInit();

GLCD\_Init();

GLCD\_Clear(White);

\_init\_box (mpool, sizeof(mpool), sizeof(U32));

tick = os\_time\_get();

os\_sys\_init(send\_task);

}

**2. Program on Linux**

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <mqueue.h>

#include <time.h>

#include <signal.h>

#include <sys/stat.h>

#include "common.h"

#define N 20 // The number of messages the sender is going to send.

#define B 1 // Number of messages the mqueue can hold.

// Time measurement variables.

struct timeval tv;

double t1;

double t2;

double t3;

int main(){

pid\_t child\_pid;

gettimeofday(&tv, NULL); // Measure time. (get t1)

t1 = tv.tv\_sec + tv.tv\_usec/1000000.0; // Convert time to seconds.

child\_pid = fork(); // Create child process.

if (child\_pid != 0) {

// Inside parent process

// Implement sender

mqd\_t qdes; // Declare message queue descriptors(handler).

char qname[] = "/msgqueue"; //Queue name.

mode\_t mode = S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IWGRP | S\_IROTH | S\_IWOTH; // Mode of file.

struct mq\_attr attr; // Queue attribute.

attr.mq\_maxmsg = B; // Set message queue size.

attr.mq\_msgsize = sizeof(int); // Set message size.

attr.mq\_flags = 0;

qdes = mq\_open(qname, O\_RDWR | O\_CREAT, mode, &attr); // Create and open the message queue.

srand(time(0));

int j = 0;

int i = 0;

// Measure time. (get t2)

gettimeofday(&tv, NULL);

t2 = tv.tv\_sec + tv.tv\_usec/1000000.0;

do {

i = rand() % 50; // Generate random number.

// Send random number.

if (attr.mq\_curmsgs < B) {

if (mq\_send(qdes, (char \*)&i, sizeof(int), 0) == -1) {

perror("mq\_send() failed");

} else {

//printf("sending %i\n", i);

j++;

}

}

} while(j < N);

wait(); // wait for child process to finish.

// Measure time. (get t3)

gettimeofday(&tv, NULL);

t3 = tv.tv\_sec + tv.tv\_usec/1000000.0;

printf("time to initialize system: %f seconds\n", (t2-t1));

printf("time to transimit data: %f seconds\n", (t3-t2));

// close message queue

if (mq\_close(qdes) == -1) {

perror("mq\_close() failed");

exit(2);

}

} else {

// Inside child process

// Implement receiver

mqd\_t qdes;

char qname[] = "/msgqueue";

mode\_t mode = S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IWGRP | S\_IROTH | S\_IWOTH;

struct mq\_attr attr;

attr.mq\_maxmsg = B;

attr.mq\_msgsize = sizeof(int);

attr.mq\_flags = 0;

qdes = mq\_open(qname, O\_RDONLY, mode, &attr);

if (qdes == -1 ) {

perror("mq\_open()");

exit(1);

}

int j = 0;

while(j < N) {

int i;

struct timespec ts = {time(0) + 5, 0};

// Receive messages.

if (mq\_timedreceive(qdes, (char \*) &i, sizeof(int), 0, &ts) == -1) {

perror("mq\_timedreceive() failed");

} else {

printf("%i is consumed.\n", i);

j++;

}

}

}

return 0;

}