Yashraj Nikam (yan7)

Dhruv Kedia (dk1157)

Project 2 Report

Implementation

Data Structures

TCB: This structure stores all the information about a thread such as the thread identifier, status, context, time quantum, the thread that it is waiting on, value pointer and return value.

worker_status: This data structure enumerates the various states a thread can be in such as ready, running, blocked et cetera.

rqueue: This is a list of all the threads prioritized by the number of quantums used up by the respective thread.

worker_mutex_t: This is the lock variable which allows mutually exclusive access to the shared resource, the 'sum' in this case.

API

Core

createTCB(): Used to create the context for a thread.

enqueue(): Add a thread to the list of threads.

dequeue(): Removes a job from a thread queue and frees the memory.

Helper

initTimer(): This method initializes the timer used for scheduling based on the time quantum value.

makeMainContext(): This is used to create an orchestration thread which waits on all the other user generated threads until they complete their execution.

findTCB(): Finds the context of a thread in the thread list.

findInMLFQ(): Finds the context of a thread in the thread list in MLFQ.

unblockThreads(): This unblocks the threads waiting on a given thread when that specific thread exits.

unblockThreadsInMlfq(): This performs the unblock threads on all the queues in MLFQ.

dequeueDestroyed(): Identifies the jobs with status as Destroy and calls dequeue on it.

dequeueDestroyedMLFQ(): Calls dequeueDestroyed on all the queues in MLFQ.

removeFromQueue(): Removes a thread from the MLFQ queue.

priorityBoost(): Increase priority after a threshold in the MLFQ scheduling policy.

reducePriority(): Reduces priority after completion of a time quantum in MLFQ.

Threads

worker_create(): It creates a new thread and performs the initialization operation by calling subroutines such as createTCB and enqueue.

worker_yield(): This function is called when a thread is supposed to voluntarily relinquish the CPU based on the scheduling policy.

worker_exit(): This function is called when a thread terminates. If the thread has a value pointer, its value is set and its status is marked as destroyed. Otherwise, we set its return value to be the valuePtr parameter passed to exit(). At the end, we unblock any threads that were waiting on this one using the unblockThreads subroutine.

worker_join(): This function checks if the thread to be joined has not finished, then we put the current thread to blocked state, give the value passed as an argument to the thread to be joined, and then invoke the scheduler to run the next thread else we check to see if the thread to be joined is already finished. If so, then we set the thread's status to destroy, indicating that we can free its memory and return any values if value_ptr is not NULL.

Mutex Functions

worker_mutex_init(): Here we simply set the mutex values to their defaults which means waitList is NULL and lock is 0.

worker_mutex_lock(): This method should acquire the mutex lock. We use atmoic_flag_test_and_set to check the mutex lock. If this is successful then we enter the critical section.

worker_mutex_unlock(): Here we run through the list of threads that are waiting for the mutex, mark their tcb status as run, and then free the entire waiting list in the mutex, so that the threads can compete for the lock again later.

worker_mutex_destroy(): Here we simply make a call to the previous unlock(), since it also handles 'destroying' the threads waiting and resets the lock.

Scheduling

sched_psjf(): Finds the next job to be executed as per PSJF policy and schedules it.

findNextJobPSJF(): Finds the next job to be executed as per PSJF policy.

sched_mlfq(): Finds the next job to be executed as per MLFQ policy and schedules it.

findNextJobMLFQ(): Finds the next job to be executed as per MLFQ policy.

Scheduling:

```
static void schedule()
{
	#ifndef MLFQ
		dequeueDestroyed(head);
		sched_psjf();
	#else
		dequeueDestroyedMLFQ();
		sched_mlfq();
	#endif
}
```

Every time the time slice is exhausted, an interrupt is generated and control is directed to our schedule() function. If the program is compiled with "SCHED=MLFQ", the MLFQ scheduler is invoked as seen. Else, PSJF is invoked. Before any one scheduler is invoked, the ready queue is searched for any threads that have their status set to DESTROY and are dequeued.

1. **Preemptive Shortest Job First (PSJF):**

```
static void sched_psjf()
{
      if(head)
      {
            current -> quantum++;
            TCB* prev = current;
            if(prev->status == RUNNING)
                  prev->status = READY;
            current = scheduleNextJobPSJF();
            if(current)
            {
                  current -> status = RUNNING;
                  tot_cntx_switches++;
                  if(current->firstScheduled == -1)
                        current -> firstScheduled = clock();
                  initTimer(QUANTUM);
                  swapcontext(&prev->ctx,&current->ctx);
            }
            else
                  current = prev;

      }
}
```

Shortest Job First requires us to know the burst times before executing a thread which is seldom the case in real world applications. So, we need to estimate the burst times before execution. In order to do this, we assume that the past is a reflection of the future and hence use the amount of time quantums used up by a thread as the determining factor in scheduling that specific thread. Lower this value, higher the chances of this thread getting scheduled. To achieve this, we maintain the value of the time quantum spent by every thread and in the method findNextJobPSJF() we traverse the thread list to find the job with the smallest value of time quantum. Then, in sched_psjf(), first a null check is performed on head. If head is NULL, the rqueue is empty and there are no further jobs to schedule. Else, a new job is scheduled. Before scheduling the new job, the quantum field of TCB of the previous thread that was running is incremented. If the previous process was not blocked or terminated, its status is changed back to READY. If the new job returned by findNextJobPSJF() is NULL the current is assigned back to

prev. Else, the newly selected thread's status is turned to RUNNING, tot_cntx_switches is incremented, timer is set and context is swapped.

2. **Multi-Level Feedback Queue Scheduling (MLFQ):**

```
static void sched_mlfq()
{
      current -> quantum++;
      TCB* prev = current;
      if(prBoostCounter >= PR_BOOST_QUANTUM)
            priorityBoost();
      else
            reducePriority();
      if(prev->status == RUNNING)
            prev->status = READY;
      int index;
      current = findNextJobMLFQ(&index);
      if(current)
      {
            current -> status = RUNNING;
            prBoostCounter += QUANTUM_MLFQ+index*QUANTUM_MLFQ;
            initTimer(QUANTUM_MLFQ+index*QUANTUM_MLFQ);
            tot_cntx_switches++;
            if(current->firstScheduled == -1)
                  current -> firstScheduled = clock();
            swapcontext(&prev->ctx,&current->ctx);
      }
      else
            current = prev;
}
```

The time slice used for every queue is calculated as `QUANTUM_MLFQ+index*QUANTUM_MLFQ` as shown in the code. For example, if QUANTUM_MLFQ is defined to be 1000ms then 1st level will have 1000ms, 2nd level will have 2000ms, 3rd will have 3000ms etc as their time slice.

The 5 rules used to implement MLFQ are:

- **Rule 1:** If Priority(A) $>$ Priority(B), A runs (B doesn't).
- **Rule 2:** If Priority(A) $=$ Priority(B), A & B run in round-robin fashion using the time slice (quantum length) of the given queue.
- **Rule 3:** When a job enters the system, it is placed at the highest priority (the topmost queue).
- **Rule 4:** Once a job uses up its time allotment at a given level (regardless of how many times it has given up the CPU), its priority is reduced (i.e., it moves down one queue).
- **Rule 5:** After some time period S, move all the jobs in the system to the topmost queue.

We will now see how each rule is implemented:

- The first 2 rules are implemented by findNextJobMLFQ(int*)

```
TCB* findNextJobMLFQ(int* index)
{
	for(int i = 0; i < NUM_OF_MLFQ; i++)
		if(multiLevelQueues[i])
		{
			if(multiLevelQueues[i] ->value -> status == READY)
			{
				*index = i;
				return multiLevelQueues[i] -> value;
			}
			else
			{
				rqueue* temp = multiLevelQueues[i];
				while(temp && temp->value->status != READY)
					temp = temp -> next;
				if(!temp)
					continue;
				*index = i;
				return temp -> value;
			}
		}
	*index = -1;
	return NULL;
```

```
}
```

For every queue, where priority of each thread in that queue is equal, findNextJobMLFQ(int*) traverses the queue until it finds a valid job in a Round Robin fashion. This function starts from the top level queue with the highest priority and works its way down until it finds a job with READY status. Priority is given to the threads in upper level queues hence satisfying the first two rules.

- The 3rd rule is implemented in the worker_create(worker_t *, pthread_attr_t * ,void*(*f)(void*), void *)

```
enqueue(&multiLevelQueues[0],new);
```

Whenever a thread is created, it is added to the first queue (multiLevelQueues[0]) of MLFQ by enqueue().

- The 4th rule is implemented in reducePriority()

```
void reducePriority()
{
	int currentIndex;
	rqueue* prev = findInMLFQ(current->threadId,&currentIndex);
	if(!prev)
		return;
	if(currentIndex == -1)
				//all jobs terminated
	{
		current = NULL;
		return;
	}
	else if(currentIndex == NUM_OF_MLFQ - 1)
	//Already in lowest queue, move to tail
	{
		rqueue* oldHead = multiLevelQueues[currentIndex];
		if(multiLevelQueues[currentIndex] == prev)
			if(multiLevelQueues[currentIndex]->next)
				multiLevelQueues[currentIndex] =
multiLevelQueues[currentIndex] -> next;
```

```
                else
                        return;
            else
            {
                rqueue* temp = oldHead;
                while(temp->next && temp->next->value->threadId !=
prev->value->threadId)
                        temp = temp -> next;
                temp -> next = prev->next;
                prev->next = NULL;
            }
            rqueue* last = oldHead;
            while(last -> next)
                last = last -> next;
            last -> next = prev;
            prev -> next = NULL;
            return;
      }
      removeFromQueue(prev,currentIndex);
      enqueue(&multiLevelQueues[currentIndex+1],prev);
      return;
}
```

There are 3 cases possible in this scenario:

1. *The thread is not present in any of the queues*: In this case we just set current to NULL and return.

2. *The thread is already at the last level of Multi-level queue:* In this case, we just move the job to tail to emulate Round Robin.

3. *The thread exists and is not present in the last level*: In this case, we invoke removeFromQueue() which removes the queue from the current level of queue. Then we invoke enqueue() and pass the head of the next level of queue and the thread to be added.

- The 5th rule is satisfied by priorityBoost()

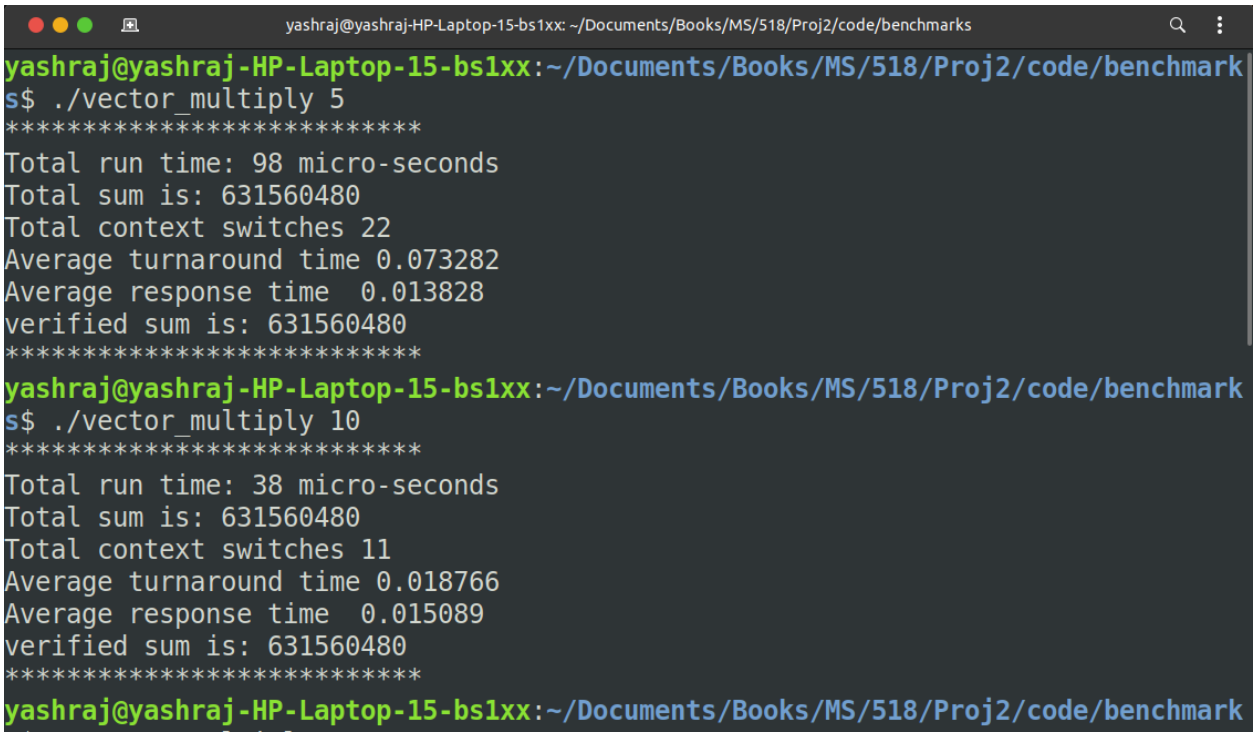
```
void priorityBoost()
{
	int i = 1;
	if(!multiLevelQueues[0])
		for(i=1;i<NUM_OF_MLFQ;i++)
			if(multiLevelQueues[i])
			{
				multiLevelQueues[0] = multiLevelQueues[i];
				multiLevelQueues[i] = NULL;
				i++;
				break;
			}
	if(i == NUM_OF_MLFQ)
		return;
	for(i; i < NUM_OF_MLFQ; i++)
	{
		if(multiLevelQueues[i])
		{
			rqueue* temp = multiLevelQueues[0];
			while(temp->next)
				temp = temp -> next;
			temp -> next = multiLevelQueues[i];
			multiLevelQueues[i] = NULL;
		}
	}
	prBoostCounter = 0;
}
```

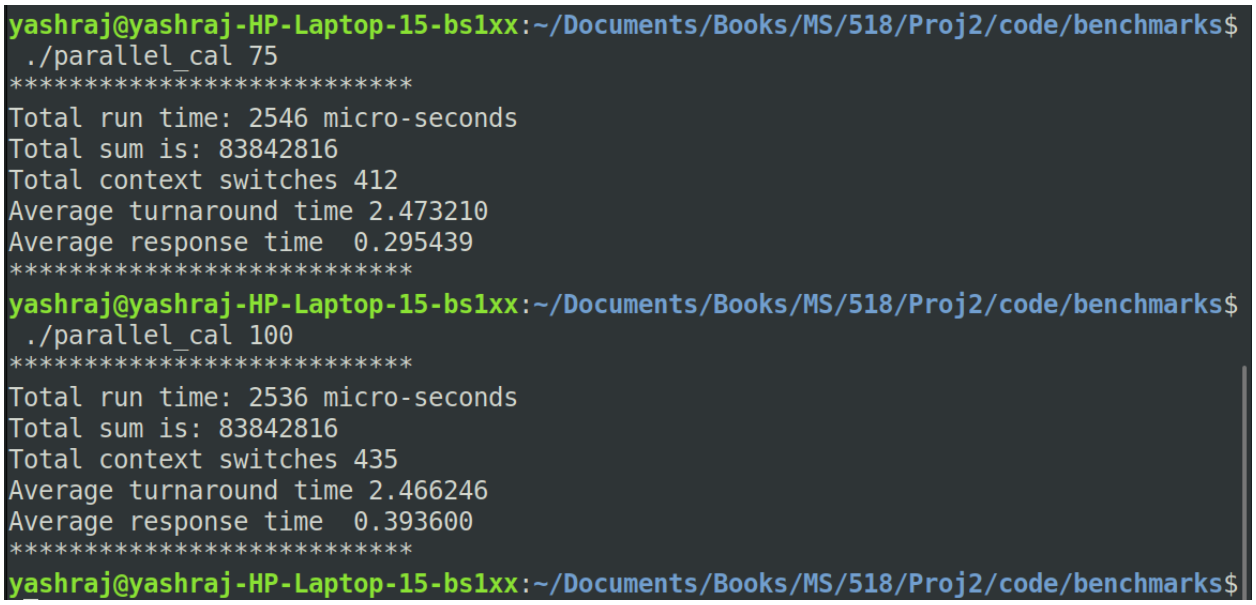
A global variable prBoostCounter is declared and initialized to 0. It is incremented everytime by the quantum calculated for the queue at that level. PR_BOOST_QUANTUM is the macro defined indicating the S value of MLFQ i.e. the time interval after which priority is to be boosted. When prBoostCounter is greater than or equal to PR_BOOST_QUANTUM, priorityBoost() is invoked. The method just simply moves all the threads to the uppermost level with most priority.

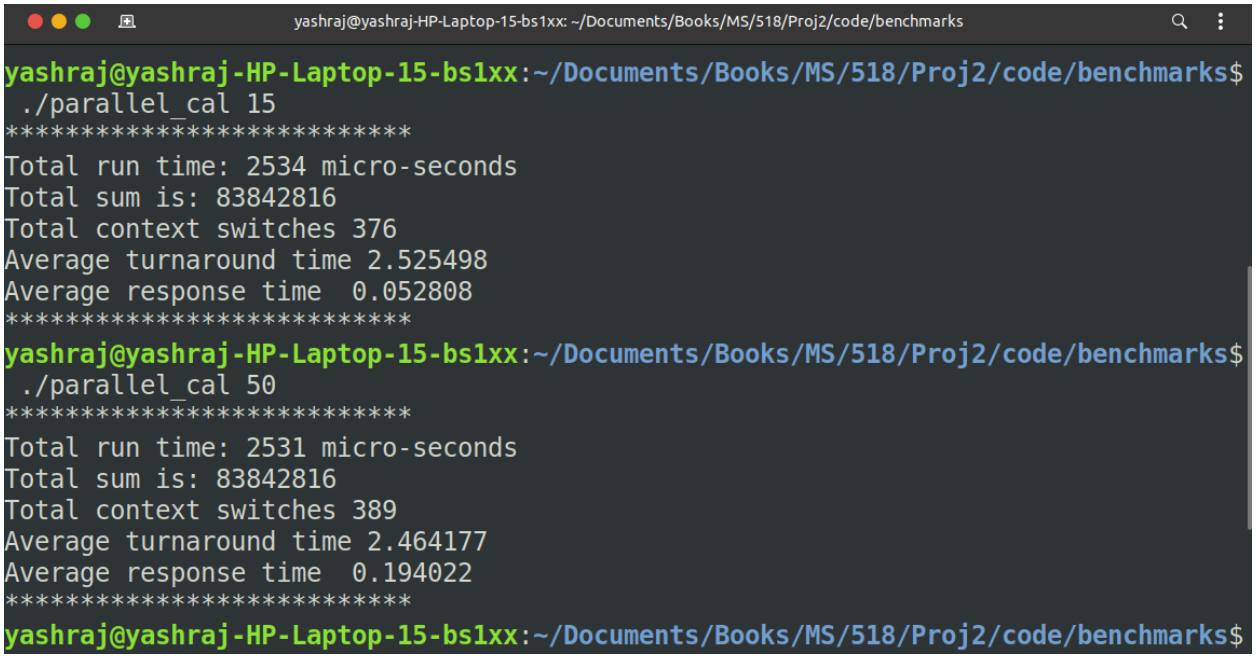
Benchmark Results for SJF

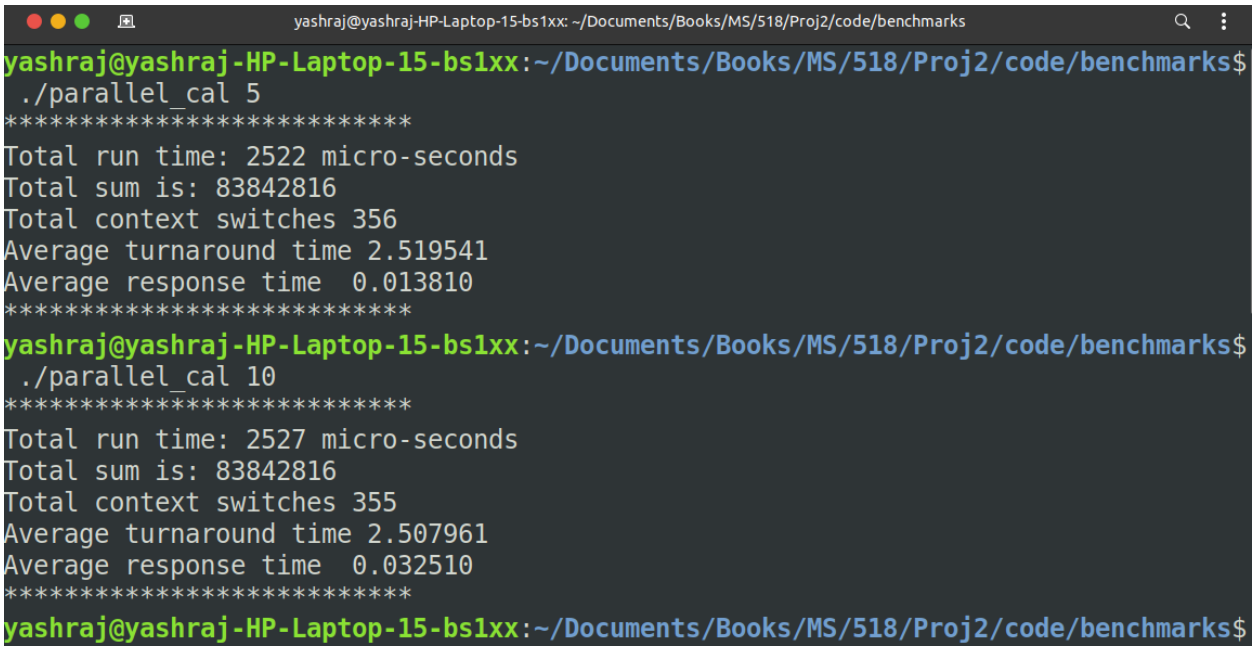
```
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmark
s$ ./vector_multiply 75
**************************
Total run time: 65 micro-seconds
Total sum is: 631560480
Total context switches 76
Average turnaround time 0.032357
Average response time  0.031497
verified sum is: 631560480
**************************
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmark
s$ ./vector_multiply 100
**************************
Total run time: 62 micro-seconds
Total sum is: 631560480
Total context switches 101
Average turnaround time 0.032173
Average response time  0.031576
verified sum is: 631560480
**************************
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmark
s$
```

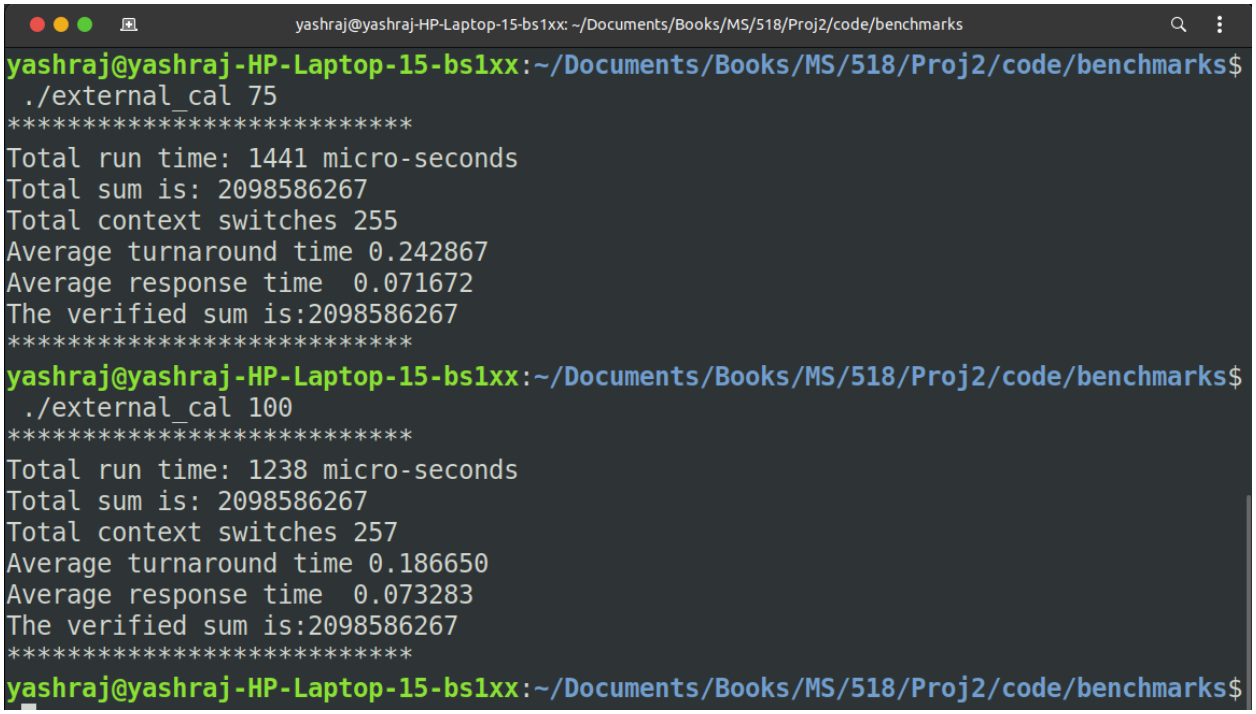
```
yashraj@yashraj-HP-Laptop-15-bs1xx: ~/Documents/Books/MS/518/Proj2/code/benchmarks
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmark
s$ ./vector_multiply 15
**************************
Total run time: 43 micro-seconds
Total sum is: 631560480
Total context switches 16
Average turnaround time 0.021520
Average response time  0.018824
verified sum is: 631560480
**************************
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmark
s$ ./vector_multiply 50
**************************
Total run time: 64 micro-seconds
Total sum is: 631560480
Total context switches 51
Average turnaround time 0.032645
Average response time  0.031383
verified sum is: 631560480
**************************
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmark
```

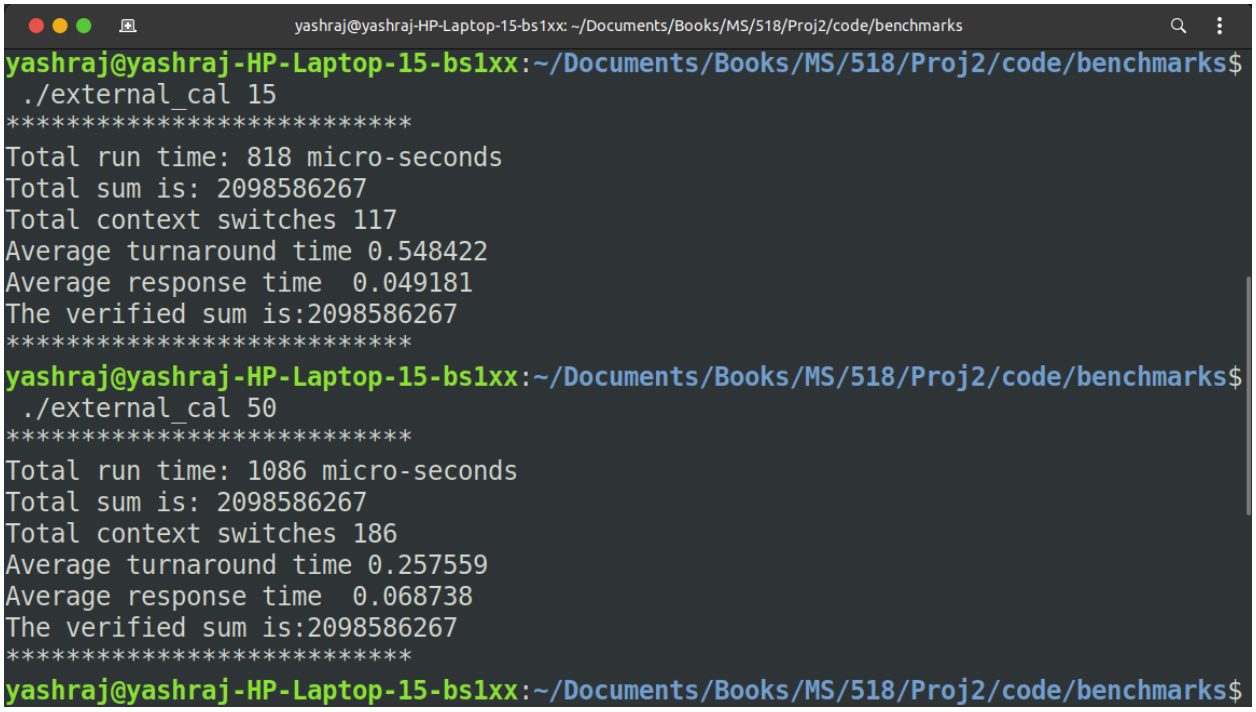

yashraj@yashraj-HP-Laptop-15-bs1xx: ~/Documents/Books/MS/518/Proj2/code/benchmarks
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./vector_multiply 5
****
Total run time: 98 micro-seconds
Total sum is: 631560480
Total context switches 22
Average turnaround time 0.073282
Average response time 0.013828
verified sum is: 631560480
****
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./vector_multiply 10
****
Total run time: 38 micro-seconds
Total sum is: 631560480
Total context switches 11
Average turnaround time 0.018766
Average response time 0.015089
verified sum is: 631560480
****
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmark

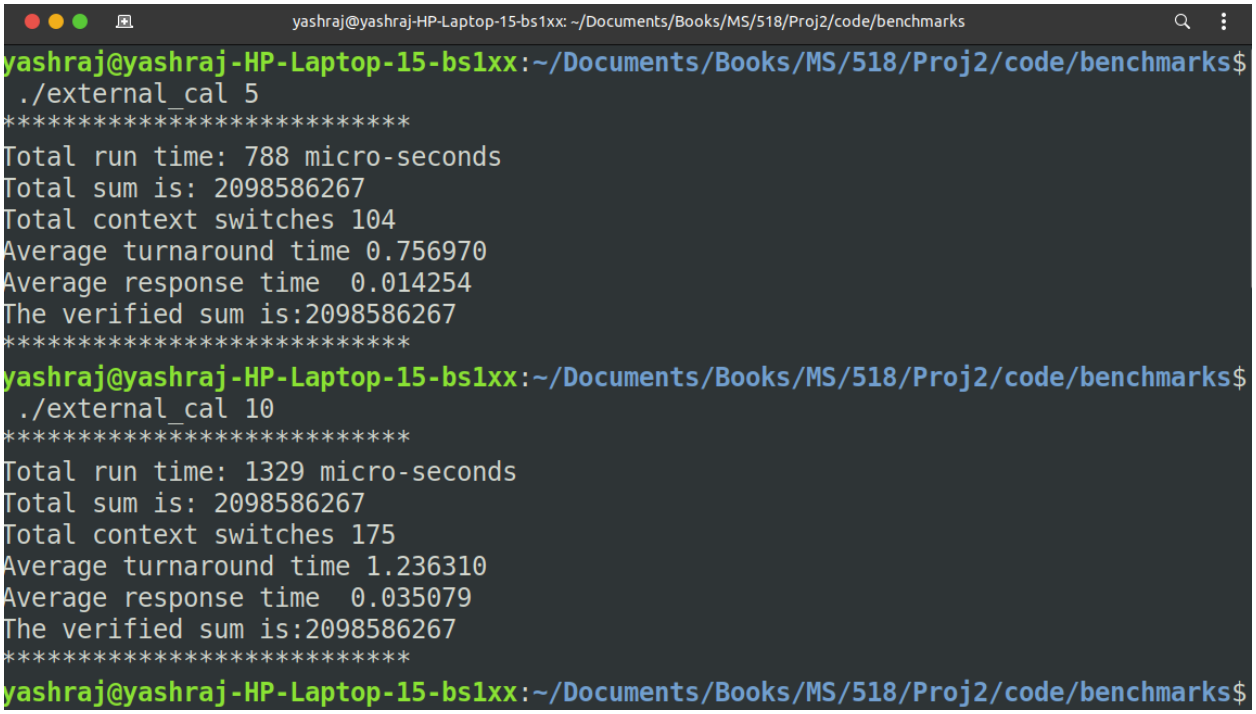

yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./parallel_cal 75
****
Total run time: 2546 micro-seconds
Total sum is: 83842816
Total context switches 412
Average turnaround time 2.473210
Average response time 0.295439
****
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./parallel_cal 100
****
Total run time: 2536 micro-seconds
Total sum is: 83842816
Total context switches 435
Average turnaround time 2.466246
Average response time 0.393600
****
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$


yashraj@yashraj-HP-Laptop-15-bs1xx: ~/Documents/Books/MS/518/Proj2/code/benchmarks
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./parallel_cal 15
****
Total run time: 2534 micro-seconds
Total sum is: 83842816
Total context switches 376
Average turnaround time 2.525498
Average response time 0.052808
****
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./parallel_cal 50
****
Total run time: 2531 micro-seconds
Total sum is: 83842816
Total context switches 389
Average turnaround time 2.464177
Average response time 0.194022
****
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$

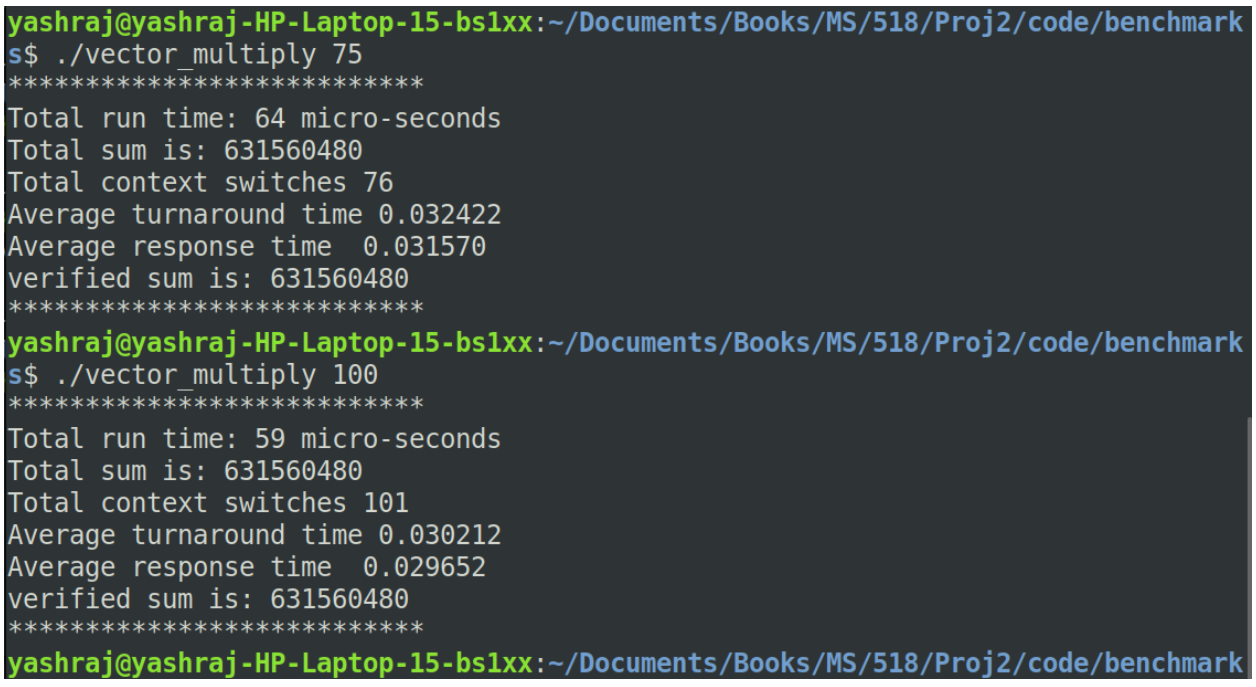

yashraj@yashraj-HP-Laptop-15-bs1xx: ~/Documents/Books/MS/518/Proj2/code/benchmarks
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./parallel_cal 5
****
Total run time: 2522 micro-seconds
Total sum is: 83842816
Total context switches 356
Average turnaround time 2.519541
Average response time 0.013810
****
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./parallel_cal 10
****
Total run time: 2527 micro-seconds
Total sum is: 83842816
Total context switches 355
Average turnaround time 2.507961
Average response time 0.032510
****
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$


yashraj@yashraj-HP-Laptop-15-bs1xx: ~/Documents/Books/MS/518/Proj2/code/benchmarks
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./external_cal 75
****
Total run time: 1441 micro-seconds
Total sum is: 2098586267
Total context switches 255
Average turnaround time 0.242867
Average response time 0.071672
The verified sum is:2098586267
****
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./external_cal 100
****
Total run time: 1238 micro-seconds
Total sum is: 2098586267
Total context switches 257
Average turnaround time 0.186650
Average response time 0.073283
The verified sum is:2098586267
****
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$

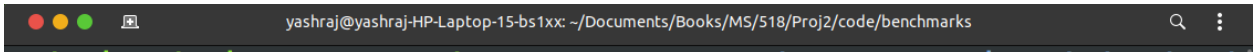

yashraj@yashraj-HP-Laptop-15-bs1xx: ~/Documents/Books/MS/518/Proj2/code/benchmarks
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./external_cal 15
****
Total run time: 818 micro-seconds
Total sum is: 2098586267
Total context switches 117
Average turnaround time 0.548422
Average response time 0.049181
The verified sum is:2098586267
****
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./external_cal 50
****
Total run time: 1086 micro-seconds
Total sum is: 2098586267
Total context switches 186
Average turnaround time 0.257559
Average response time 0.068738
The verified sum is:2098586267
****
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$



Benchmark results for MLFQ



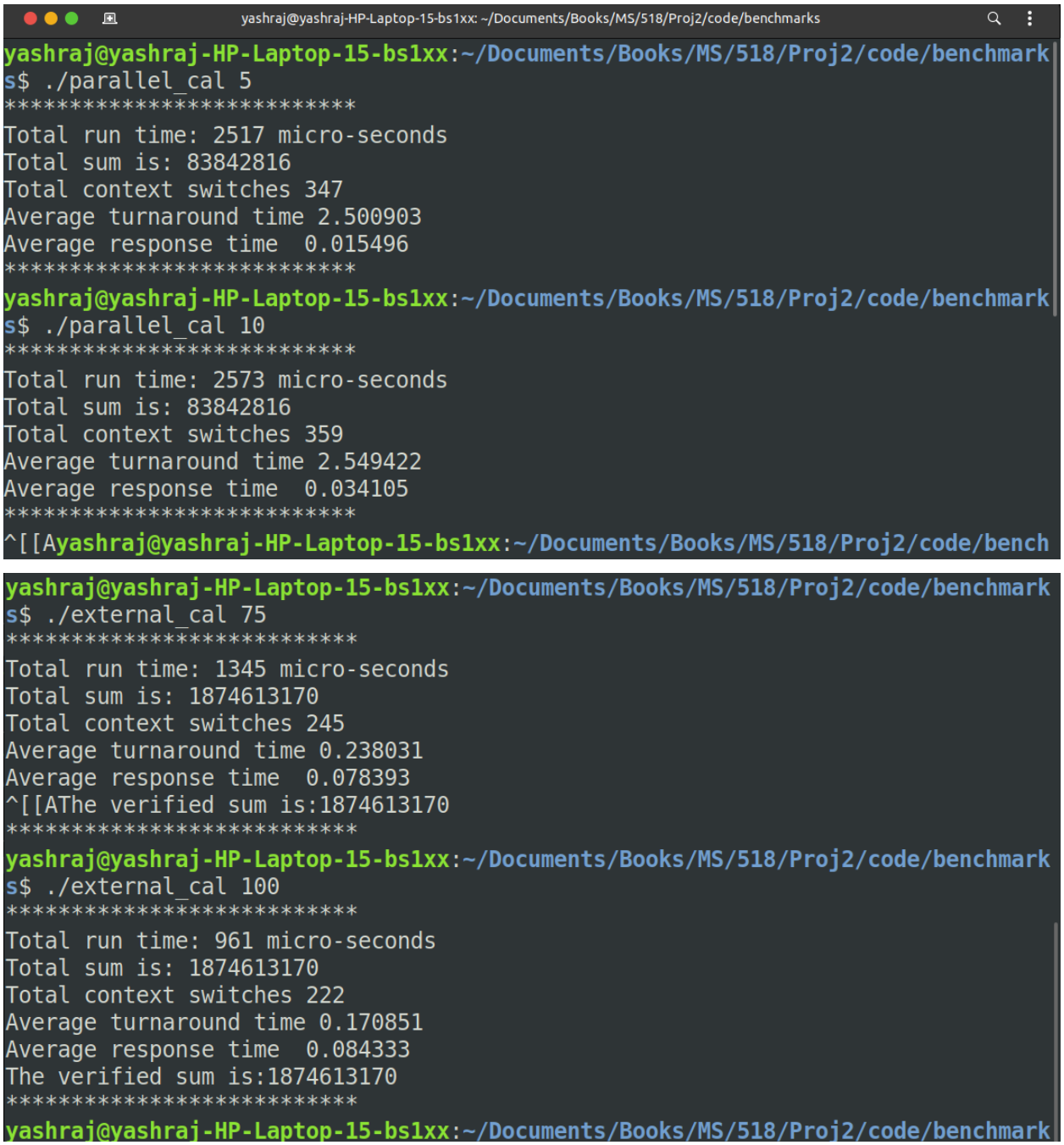
```
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmark
s$ ./vector_multiply 15
**************************
Total run time: 39 micro-seconds
Total sum is: 631560480
Total context switches 16
Average turnaround time 0.019401
Average response time  0.016863
verified sum is: 631560480
**************************
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmark
s$ ./vector_multiply 50
**************************
Total run time: 63 micro-seconds
Total sum is: 631560480
Total context switches 51
Average turnaround time 0.032461
Average response time  0.031227
verified sum is: 631560480
**************************
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmark
```

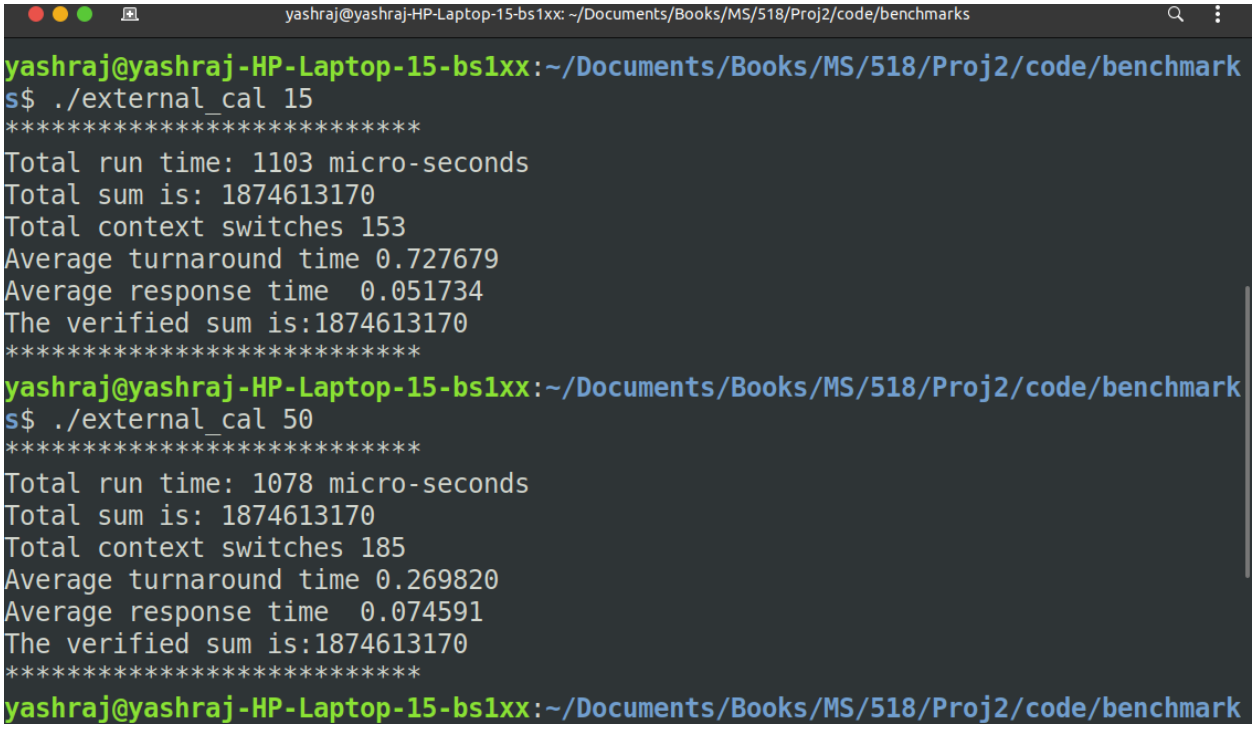


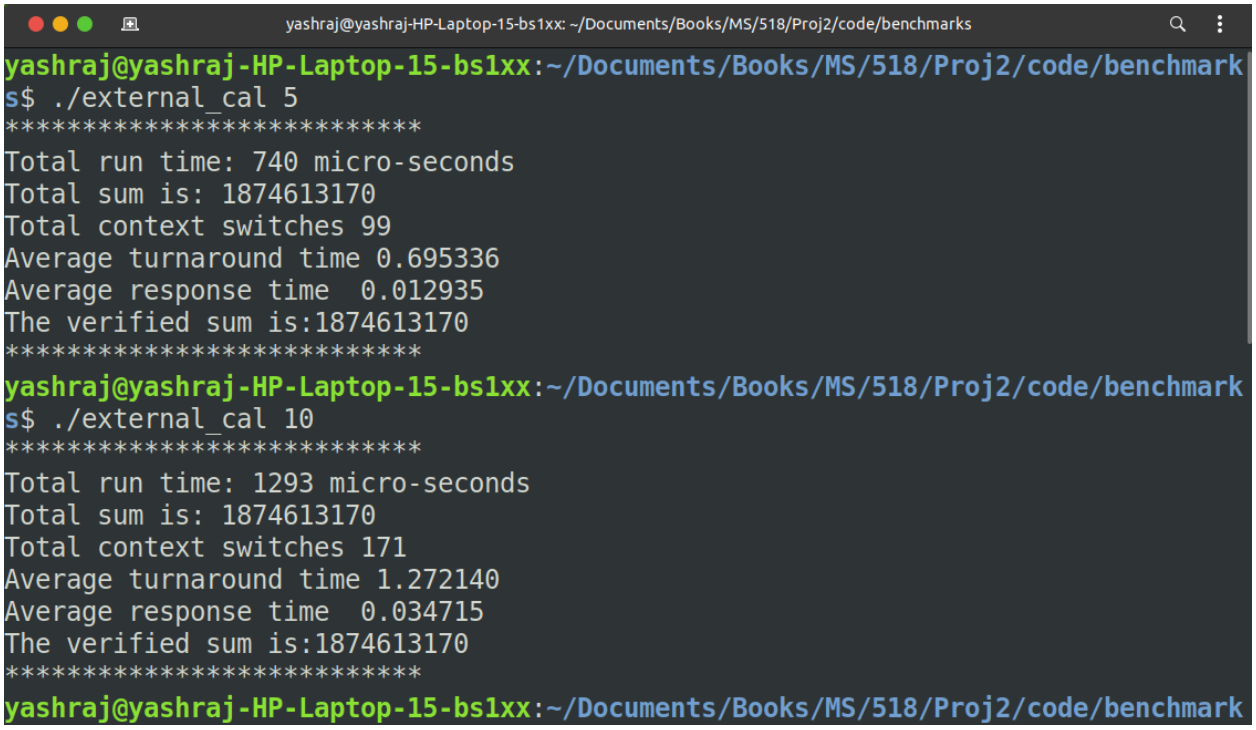

```
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmark
s$ ./vector_multiply 5
**************************
Total run time: 98 micro-seconds
Total sum is: 631560480
Total context switches 20
Average turnaround time 0.067254
Average response time  0.015407
verified sum is: 631560480
**************************
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmark
s$ ./vector_multiply 10
**************************
Total run time: 39 micro-seconds
Total sum is: 631560480
Total context switches 11
Average turnaround time 0.021254
Average response time  0.017688
verified sum is: 631560480
**************************
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmark
```

```
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmark
s$ ./parallel_cal 75
****************************
Total run time: 2569 micro-seconds
Total sum is: 83842816
Total context switches 414
Average turnaround time 2.374899
Average response time  0.320304
****************************
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmark
s$ ./parallel_cal 100
****************************
Total run time: 2584 micro-seconds
Total sum is: 83842816
Total context switches 434
Average turnaround time 2.385101
Average response time  0.433022
****************************
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmark
```

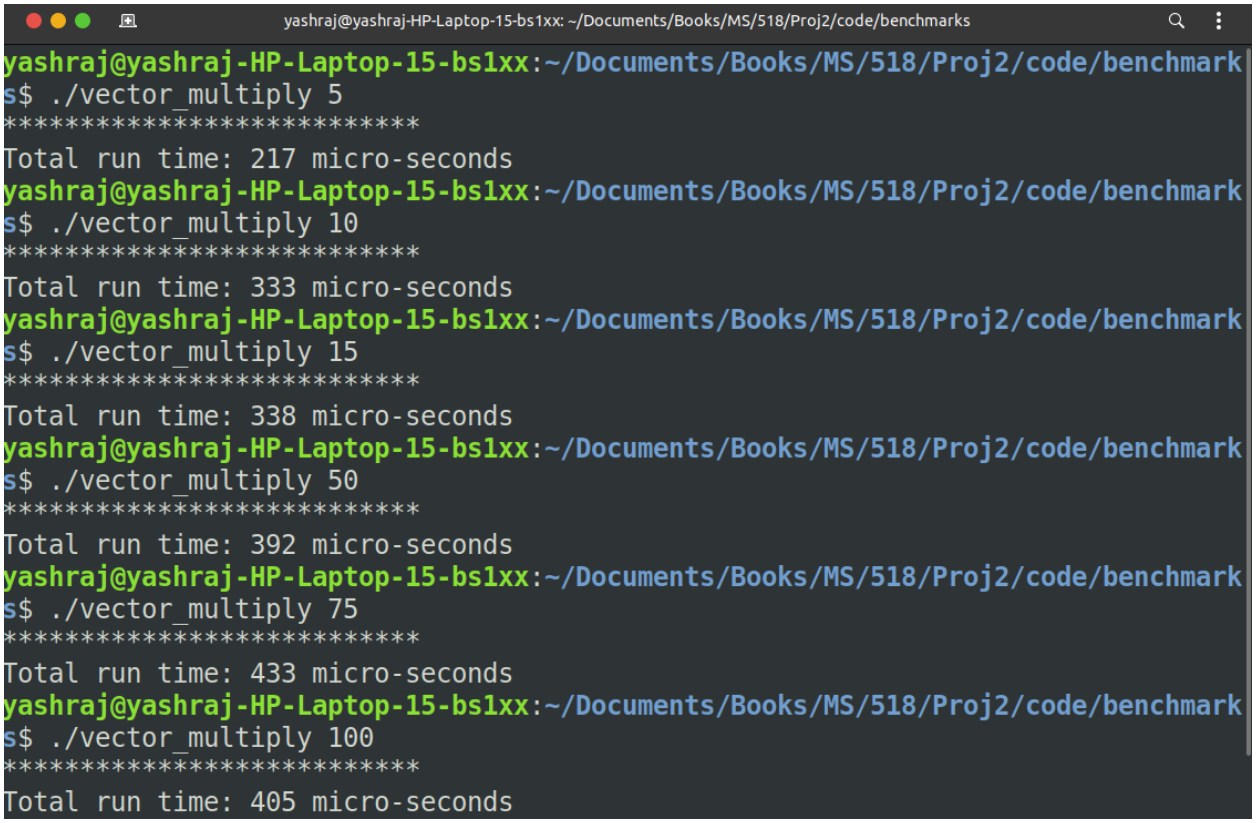
```
yashraj@yashraj-HP-Laptop-15-bs1xx: ~/Documents/Books/MS/518/Proj2/code/benchmarks
^[[Ayashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/bench
marks$ ./parallel_cal 15
****************************
Total run time: 2559 micro-seconds
Total sum is: 83842816
Total context switches 363
Average turnaround time 2.403857
Average response time  0.056570
****************************
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmark
s$ ./parallel_cal 50
****************************
Total run time: 2552 micro-seconds
Total sum is: 83842816
Total context switches 387
Average turnaround time 2.408834
Average response time  0.212419
****************************
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmark
```

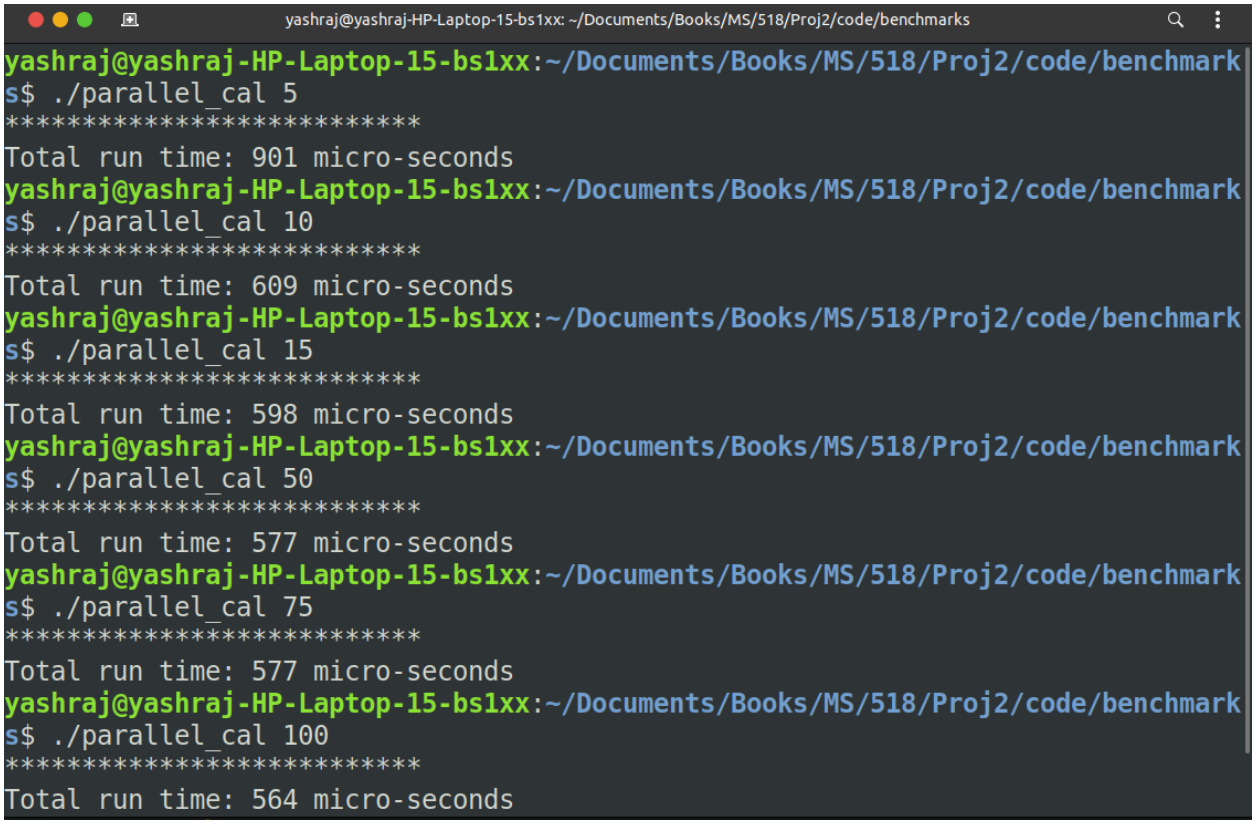

yashraj@yashraj-HP-Laptop-15-bs1xx: ~/Documents/Books/MS/518/Proj2/code/benchmarks
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./parallel_cal 5
****
Total run time: 2517 micro-seconds
Total sum is: 83842816
Total context switches 347
Average turnaround time 2.500903
Average response time 0.015496
****
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./parallel_cal 10
****
Total run time: 2573 micro-seconds
Total sum is: 83842816
Total context switches 359
Average turnaround time 2.549422
Average response time 0.034105
****
^[[Ayashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/bench
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./external_cal 75
****
Total run time: 1345 micro-seconds
Total sum is: 1874613170
Total context switches 245
Average turnaround time 0.238031
Average response time 0.078393
^[[AThe verified sum is:1874613170
****
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./external_cal 100
****
Total run time: 961 micro-seconds
Total sum is: 1874613170
Total context switches 222
Average turnaround time 0.170851
Average response time 0.084333
The verified sum is:1874613170
****
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks

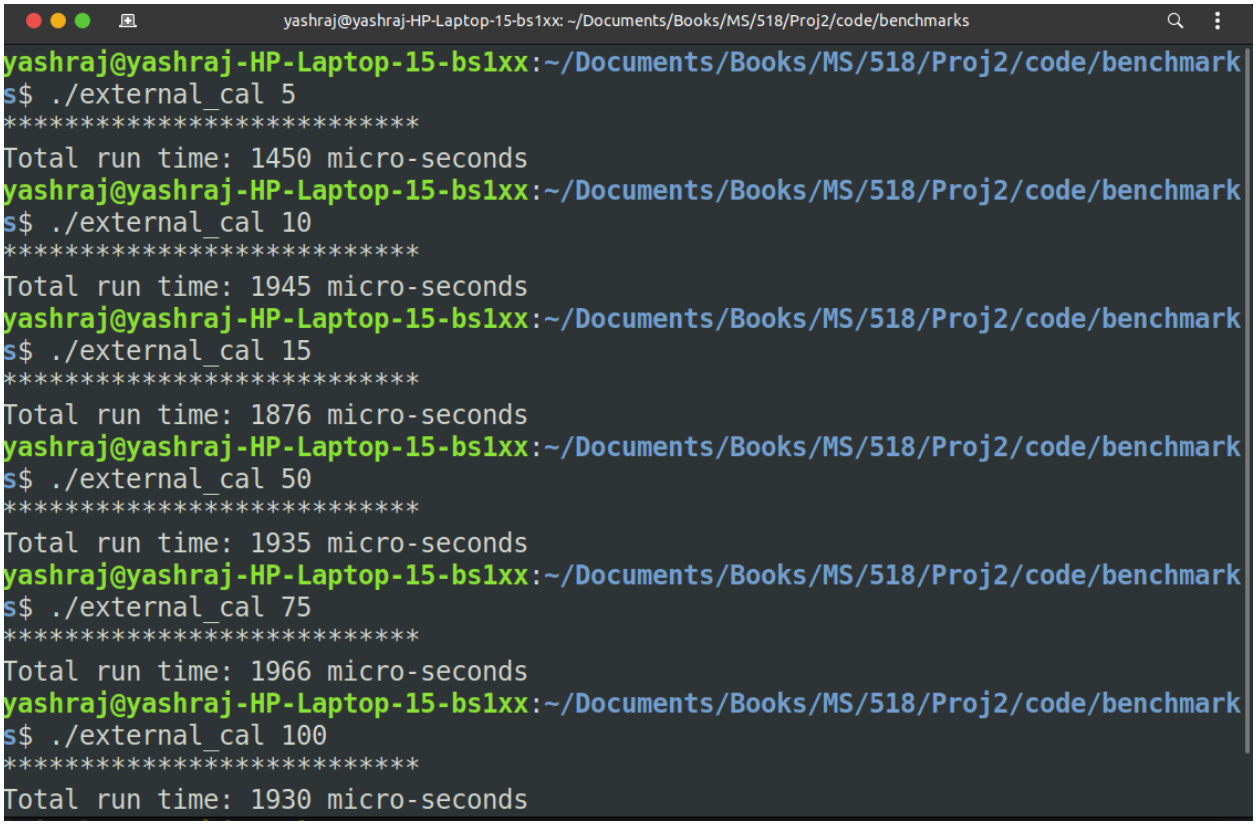

yashraj@yashraj-HP-Laptop-15-bs1xx: ~/Documents/Books/MS/518/Proj2/code/benchmarks
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./external_cal 15
****
Total run time: 1103 micro-seconds
Total sum is: 1874613170
Total context switches 153
Average turnaround time 0.727679
Average response time 0.051734
The verified sum is:1874613170
****
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./external_cal 50
****
Total run time: 1078 micro-seconds
Total sum is: 1874613170
Total context switches 185
Average turnaround time 0.269820
Average response time 0.074591
The verified sum is:1874613170
****
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmark


yashraj@yashraj-HP-Laptop-15-bs1xx: ~/Documents/Books/MS/518/Proj2/code/benchmarks
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./external_cal 5
****
Total run time: 740 micro-seconds
Total sum is: 1874613170
Total context switches 99
Average turnaround time 0.695336
Average response time 0.012935
The verified sum is:1874613170
****
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./external_cal 10
****
Total run time: 1293 micro-seconds
Total sum is: 1874613170
Total context switches 171
Average turnaround time 1.272140
Average response time 0.034715
The verified sum is:1874613170
****
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmark

Comparison with pthread




yashraj@yashraj-HP-Laptop-15-bs1xx: ~/Documents/Books/MS/518/Proj2/code/benchmarks
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./parallel_cal 5
****
Total run time: 901 micro-seconds
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./parallel_cal 10
****
Total run time: 609 micro-seconds
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./parallel_cal 15
****
Total run time: 598 micro-seconds
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./parallel_cal 50
****
Total run time: 577 micro-seconds
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./parallel_cal 75
****
Total run time: 577 micro-seconds
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./parallel_cal 100
****
Total run time: 564 micro-seconds


yashraj@yashraj-HP-Laptop-15-bs1xx: ~/Documents/Books/MS/518/Proj2/code/benchmarks
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./external_cal 5
****
Total run time: 1450 micro-seconds
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./external_cal 10
****
Total run time: 1945 micro-seconds
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./external_cal 15
****
Total run time: 1876 micro-seconds
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./external_cal 50
****
Total run time: 1935 micro-seconds
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./external_cal 75
****
Total run time: 1966 micro-seconds
yashraj@yashraj-HP-Laptop-15-bs1xx:~/Documents/Books/MS/518/Proj2/code/benchmarks$ ./external_cal 100
****
Total run time: 1930 micro-seconds

We can see that the average TAT for MLFQ is greater than SJF and the average RT for MLFQ is lower than SJF.

The total runtime for our implementation is comparatively lower than that of linux pthread.