Index

Abstract

This project presents a dynamic progress bar with pre-emptive scheduling and thread management, offering practical insights into concurrent process management.

Preemptive scheduling refers to a method where the operating system can interrupt ongoing tasks to prioritise and manage the execution of multiple processes efficiently. In this context, it likely means that the progress bar manages and updates multiple tasks concurrently, allowing for smooth and efficient task allocation and execution.

Threads are individual sequences of programmed instructions within a process. Effective thread management involves handling these threads efficiently, possibly by allocating system resources, managing their execution, and ensuring synchronisation where needed. The project likely involves managing threads to update the progress bar and execute tasks concurrently.

Applications:
1. Data Transfer Visualisations: Showing the progress of data transfers in applications like file downloads, uploads, or data synchronisation.
2. Package Management: Displaying the progress of package installation, updates, or downloads in software repositories or package managers.
3. Education: Providing a user-friendly interface for learning and understanding concurrent process management in real-time.

Objective

Develop a visual progress bar application that simulates multi-process execution using pre-emptive scheduling. Implement context switching to manage the time allocation for each process, ensuring fair and equitable CPU time distribution. Create an intuitive visualisation that accurately represents the progress of individual processes. Offer adjustable time quantum settings, allowing users to explore the impact of different scheduling parameters on process execution.

This application will have a graphical user interface (GUI) that displays multiple progress bars, each representing an individual process. The progress bars will dynamically update to reflect the progress of each process.

The application will simulate multiple processes running concurrently. These processes could represent various tasks or operations that need to be executed simultaneously.

Preemptive scheduling involves interrupting ongoing processes to allocate CPU time to other processes, ensuring fair distribution of CPU resources. Context switching is the mechanism that facilitates this interruption, allowing the application to switch between executing processes.

The progress bars will accurately represent the progress of each process. As the processes execute, their respective progress bars will update in real-time, visually indicating the portion of the task completed.

In summary, this project involves creating an interactive application that visually represents multi-process execution with preemptive scheduling. It provides users with a hands-on experience to observe and understand how different scheduling parameters influence the execution of concurrent processes. The adjustable settings add an exploratory aspect, allowing users to experiment and gain insights into preemptive scheduling algorithms' behaviours.

Introduction

This project introduces a dynamic progress bar implementation that utilises pre-emptive scheduling and thread management concepts, with potential applications in data transfer visualisations and package management.
In today's computing landscape, understanding how concurrent processes are managed is crucial. This project aims to shed light on critical concepts like context switching, multithreading, thread scheduling, and preemptive scheduling. By showcasing these concepts through the dynamic progress bar, users can practically grasp how CPU time is fairly allocated among various tasks. This knowledge becomes instrumental, especially in scenarios like data transfer visualisation and package management, where efficient concurrent data handling and synchronisation are key.
Displaying progress visually helps users comprehend ongoing data transfers like file downloads or uploads, enhancing user experience and understanding.
Visualising the progress of package installation or updates aids in managing multiple software components concurrently.
The user-friendly interface acts as a bridge between theoretical knowledge and practical application. It allows users to observe and interact with concurrent processes, facilitating a deeper understanding of how these processes are managed in real-time scenarios.

This project's impact is twofold—it serves as an educational tool, fostering a better understanding of complex computing concepts, and as a practical utility, aiding developers in optimising resource management in real-world applications.

In summary, this project's emphasis on visualising concurrent process management through a dynamic progress bar serves as a valuable educational and practical tool, bridging the gap between theoretical knowledge and real-world application in the realm of operating systems and resource management.

Hardware/Software Requirements

1. Hardware Requirements-

- Computer
- Standard multiprocessor
- Sufficient Memory

2. Software Requirements-

- An operating System
- C Development Environment

Concepts Used

1. Dynamic Progress Bar Module: Central component displaying real-time progress based on concurrent processes.

2. Pre-emptive Scheduling Module: Manages interruption and switching between tasks to ensure fair CPU time allocation.

3. Thread Management Module: Controls the creation, execution, and synchronisation of threads within processes.

4. Data Transfer Visualisation Module: Utilises the dynamic progress bar for visualising concurrent data transfer processes.

5. Package Management Module: Applies the dynamic progress bar to showcase concurrent handling and synchronisation in package management.

6. User Interface Module: Provides a user-friendly interface for real-time exploration of scheduling strategies.

7. Educational Context Integration: Connects to operating systems courses, enhancing learning in context switching, multithreading, and thread scheduling.

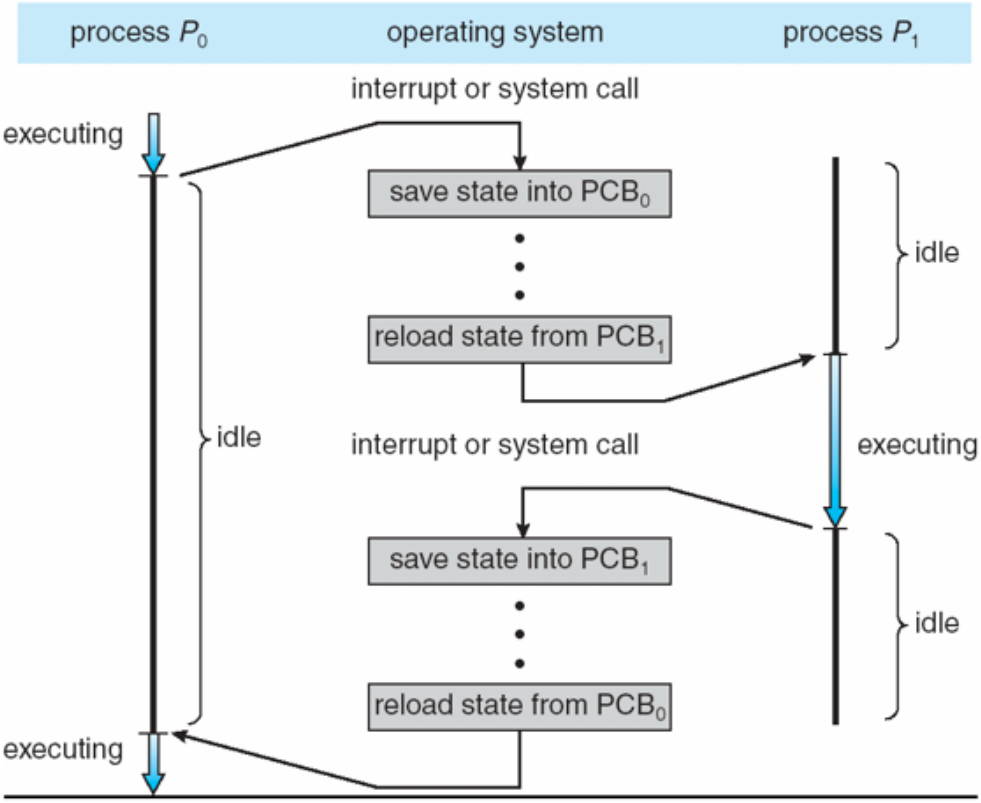
8. Practical Application Integration:Adaptable to real-world scenarios, aiding software developers in optimising resource management.

Program

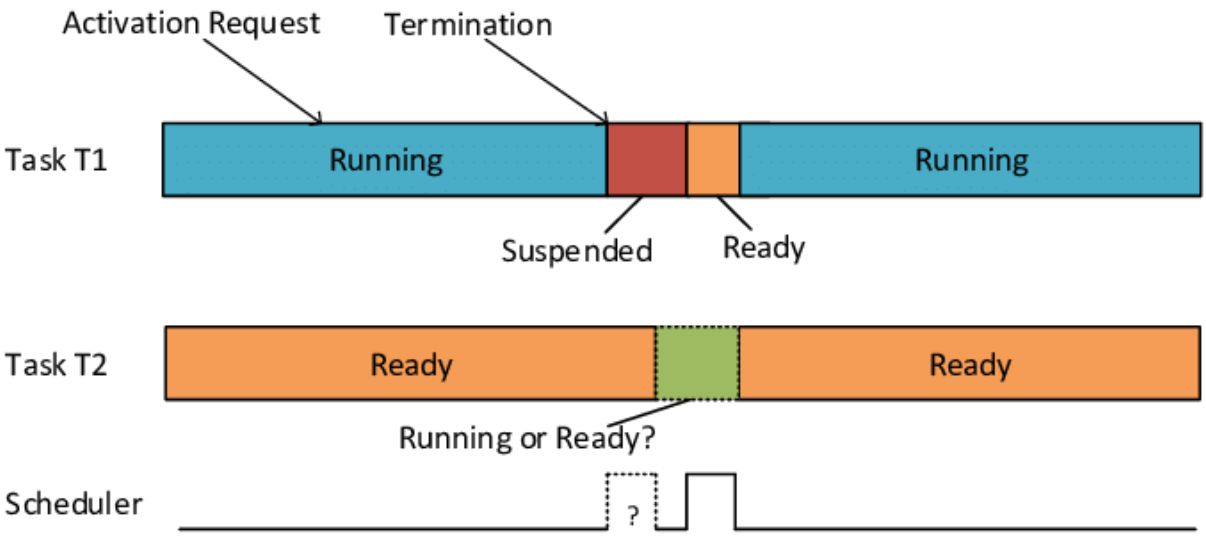
```
#include <stdio.h>
#include <unistd.h>
#include <stdbool.h>
#include <stdlib.h>
#include <pthread.h>
#define ESC "\033"
#define CSI "["
#define PREV_LINE "F"
#define BACKSPACE "D"
const int PROG_BAR_LEN = 30;
const int NUM_THREADS = 5;
typedef struct {
int count_to_val;
int progress;
pthread_t thethread;
} thread_info;
void update_bar(thread_info *tinfo) {
int num_chars = (tinfo->progress * 100 / tinfo->count_to_val) * PROG_BAR_LEN / 100;
printf("[");
for (int i = 0; i < num_chars; i++) {
printf("=");
}
if (tinfo->progress < tinfo->count_to_val) {
printf(ESC CSI BACKSPACE ">");
}
for (int i = num_chars; i < PROG_BAR_LEN; i++) {
printf(" ");
}
printf("]\n");
}
void *update_progress(void *arg) {
thread_info *tinfo = (thread_info *) arg;
for (int i = 0; i < tinfo->count_to_val; i++) {
tinfo->progress++;
usleep(1000);
}
pthread_exit(NULL);
}
int main(int argc, char *argv[]) {
thread_info threads[NUM_THREADS];
int total = 53142;
printf("Total: %d\n", total);
for (int i = 0; i < NUM_THREADS-1; i++) {
threads[i].count_to_val = total / NUM_THREADS;
threads[i].progress = 0;pthread_create(&threads[i].thethread, NULL, update_progress, &threads[i]);
}
threads[NUM_THREADS-1].count_to_val = total-(total/
NUM_THREADS)*(NUM_THREADS-1);
threads[NUM_THREADS-1].progress = 0;
pthread_create(&threads[NUM_THREADS-1].thethread, NULL, update_progress,
&threads[NUM_THREADS-1]);
int current_thread = 0;
```

```
bool done = false;
while (!done) {
done = true;
for (int i = 0; i < NUM_THREADS; i++) {
if (threads[i].progress < threads[i].count_to_val) {
done = false;
}
update_bar(&threads[i]);
printf("Thread %ld: %d\n", threads[i].thethread, threads[i].progress);
}
if (!done) {
printf(ESC CSI "%d" PREV_LINE, 2*(NUM_THREADS));
}
usleep(10000);
}
pthread_exit(NULL);
}
```

Architecture/Flowchart

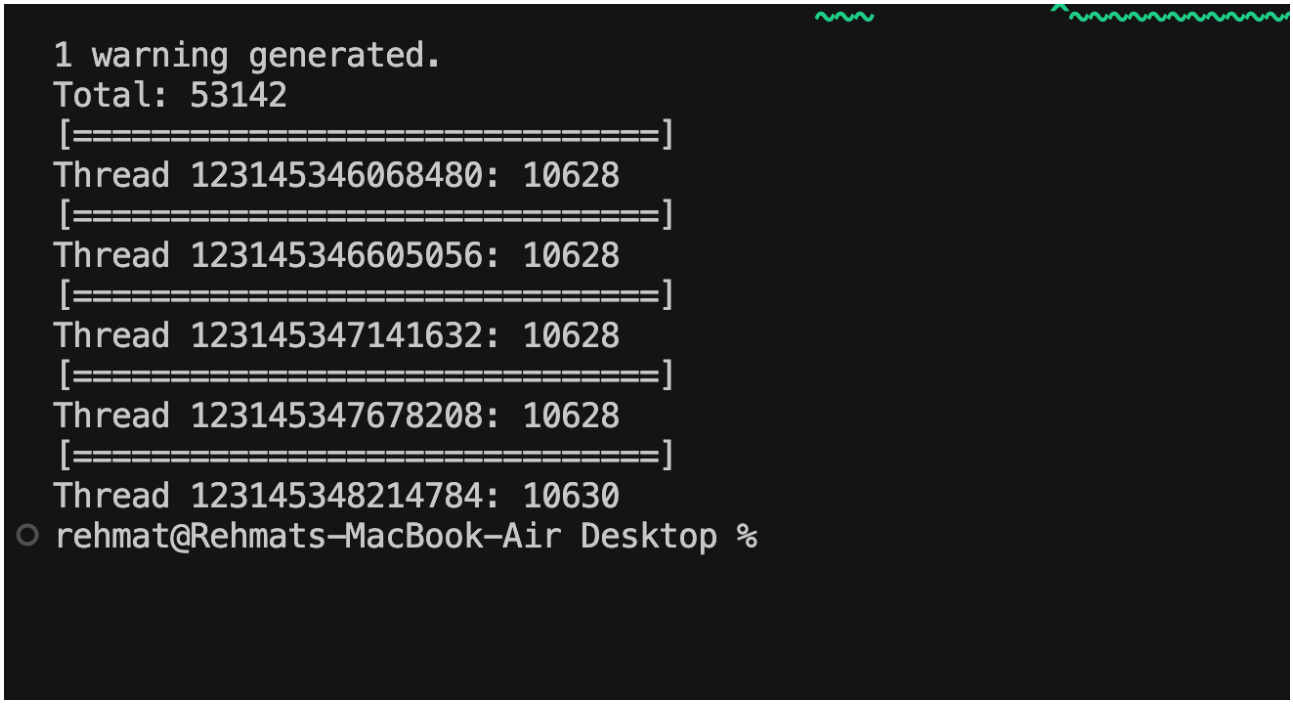


Context Switching Actual Process



Pre-emptive Scheduling

Output


1 warning generated.
Total: 53142
[==============================]
Thread 123145346068480: 10628
[==============================]
Thread 123145346605056: 10628
[==============================]
Thread 123145347141632: 10628
[==============================]
Thread 123145347678208: 10628
[==============================]
Thread 123145348214784: 10630
rehmat@Rehmats-MacBook-Air Desktop %

Conclusion

The project effectively demonstrates how multiple processes can execute concurrently within an operating system. By utilising preemptive scheduling and context switching, it illustrates how tasks can share CPU resources and run simultaneously.
The use of a visual representation, such as progress bars or graphical indicators, allows users to see how different tasks progress concurrently. This visualisation enhances understanding by presenting complex scheduling and multitasking concepts in an accessible manner.
This simulation serves as an educational tool to illustrate fundamental scheduling principles in operating systems. Students and learners can interact with the simulation to comprehend how preemptive scheduling and context switching influence the execution of multiple processes.
In software development, this simulation can be employed as a testing ground to evaluate and optimise process management algorithms. Developers can use it to experiment with different scheduling strategies and observe their impact on task execution and system performance.
The project's simulation of process execution can be extended to create monitoring tools for real-time systems. By visualising how resources are allocated and managed among multiple tasks, it can assist in monitoring system performance and identifying potential bottlenecks.
Beyond education and software development, this simulation has practical implications. It can be applied in real-world scenarios to optimise resource allocation in systems handling multiple concurrent tasks, improving overall efficiency and performance.
The project's success lies in providing a tangible representation of complex system dynamics. It allows users to observe and comprehend the behaviour of concurrent processes, aiding in the understanding of how scheduling decisions affect task execution and resource utilisation.

In summary, this project achieves its goal by effectively simulating concurrent process execution using preemptive scheduling and context switching. It not only serves educational purposes but also holds practical value in software development and real-time systems by offering insights into scheduling principles and aiding in the optimisation of process management algorithms and resource allocation.