Research on Embedded Multi-task Balancing Scheduling Algorithm

Han Zhuangzhi¹, Ma Tianlin²

¹Information and Communication Engineering, Shijiazhuang Campus, Army Engineering University, No. 97 Heping West Road, Shijiazhuang City, Hebei Province, China

²Information and Communication Engineering, Shijiazhuang Campus, Army Engineering University, No. 97 Heping West Road, Shijiazhuang City, Hebei Province, China

Abstract. For embedded systems, there are two cases of using an operating system and not using an operating system. When the real-time task is accomplished by the embedded system of the operating system, the task needs to meet certain conditions and occupy part of the processor's resources. Therefore, based on the method of event interruption, timed interruption and task decomposition, the real-time performance of the completion of the task of the embedded system is achieved. Finally, an embedded radar track compression scheduling algorithm is designed. It is proved through experiment that the track data can be compressed and transmitted in real time.

1 INTRODUCTION

The problem of multitasking processing lies mainly in the scheduling among different tasks [1]. According to the characteristics of the task and the requirements for completing the task, the method of scheduling is also different [2]. According to the nature of the task can be divided into static scheduling and dynamic scheduling. Static scheduling assigns priorities to tasks based on the static attributes of tasks, which can reduce the storage and computational costs, but has poor flexibility [3]. Dynamic Scheduling According to the priority of dynamic task assignment during task execution, EDF (Earliest Deadline First) is a typical dynamic scheduling algorithm that schedules tasks based on task deadline [4]. Its advantage is that it can be performed according to the task. Dynamic execution of tasks requires the use of resources such as processor memory and computational resources. According to the real-time requirements of task completion, it can be divided into hard real-time scheduling and soft real-time scheduling. Hard real-time scheduling is used for tasks with high real-time requirements. Immediate processing is required when tasks occur. Soft real-time scheduling can delay task execution time appropriately when the load is heavy. According to whether the task is preempted during execution, scheduling can be divided into preemptive scheduling and non-preemptive scheduling. Preemptive scheduling means that during the execution of a task, if a task with a higher priority needs to be executed, the processor will suspend the current task and execute the task with a higher priority [5]. The flexibility is high, but the switch of the task will occupy the resources of the processor. For non-preemptive scheduling algorithms,

they will not be preempted during the execution of the task. Relative to preemptive tasks, its real-time performance has certain limitations. The current scheduling algorithms are mostly used directly in the operating system, but the use of the operating system often requires tasks to meet certain requirements in terms of periodicity, execution time, etc. [6], and the operation of the operating system itself also requires a certain amount of processor resources. . This also limits the application of real-time operating systems in some projects. By analyzing the method of task scheduling in the operating system, it is directly applied in the embedded system to achieve real-time balanced scheduling of tasks. And it is used in target track compression. Experiments have proved that real-time compression of the target track can be achieved.

2 REAL-TIME BALANCED SCHEDULING ALGORITHM

For embedded program tasks, it is mainly divided into periodic tasks, acyclic tasks, and auxiliary tasks. The significance of scheduling is to equalize periodic tasks while ensuring real-time completion of real-time tasks.

2.1 Task Equalization

Equalization of tasks means that the tasks are divided equally in time to avoid the occurrence of tasks and bursts of computation. Tasks can be divided into two types, cycle and event.

Cyclic tasks are executed in the order of tasks. If a task is not completed, the next task will not be executed.

Unbalanced tasks will result in the limited operational efficiency of the program. The task in the loop is an executable task that is not real-time and can be used to perform tasks in a circular schedule. For some unpredictable execution time tasks, event interrupt scheduling is used. Avoid wasting processor resources due to waiting tasks. For a periodic task, the task is divided equally over the entire period, avoiding that the calculation of the task is performed only at a certain moment in the processor, while the resources of the processor are relatively left at other times. Therefore, a recursive algorithm is used to perform periodic tasks so that the computational load is equalized throughout the entire cycle.

The processor can only be executed serially during the execution of the task. In order to improve the resource utilization efficiency of the processor, it is necessary to reduce the delay between tasks and tasks [7]. Task decomposition can be used to decompose complex tasks into simple tasks, as shown in Fig. 1. Divide these simple tasks into loops and interrupt tasks. For real-time periodic tasks, tasks that are in urgent need of processing are completed during the interruption, to avoid waste of resources and time delays caused by calls between tasks. Fig. 2 shows the scheduling structure of the task.

Fig. 1. Decomposition structure of the task

Fig. 2. Task Scheduling Structure

From Fig. 1 we can see that the processor needs to complete Task1, Task2, Task3...Taskn a total of n tasks. Among the complex tasks are Task2 and Task3. And the execution time of these two tasks is unpredictable. In order to increase the utilization efficiency of the processor, these two tasks are set as interruption events. The effective allocation of single-core processors over time is completed [8]. For a complex task Task2, it can be decomposed into m tasks for Task2.1, Task2.2...Task2.m. Task2.3...Task2.m can be executed in round robin scheduling. Put it in the loop event, put Task2.1, Task2.2 in the interrupt event, complete the decomposition of the complex task. The decomposition in task Task3 is similar to the task Task2 decomposition.

2.2 Real-time Mission

One of the advantages of an embedded system is its ability to complete tasks in real time. The real-time execution of tasks is mainly accomplished through event interrupts. For tasks that require high real-time performance such as data acquisition and data reception, hard real-time interrupts are required. The deadline for the task will lead to data receiving errors.

During the execution of a cyclic task, considering that multiple interrupt events may occur at the same time, it is necessary to set priorities for all interrupt events. According to the execution time, real-time and urgency of the event, the priority of the interrupt event is set. For tasks with higher real-time requirements, such as Task2.1 and Task2.2, real-time processing is required. For non-emergency interrupt events, such as Task2.3...Task2.m, flag bits are set in the interrupt to pass through the main program. Query flags to perform tasks.

For real-time periodic tasks, such as data acquisition, data reception, etc., there is a certain gap between the received data, so the task is set to hard real-time interrupts. The data needs to be processed after receiving the data. Decompose complex data processing into simple tasks. If the interrupt task is not processed when the next interrupt arrives, data reception will be lost. As shown in Fig. 1. The tasks that need to be processed after receiving are placed in interrupts, such as Task2.1, Task2.2, etc., and the processing time is not guaranteed to be greater than the interval between two interruptions. That is, the real-time nature of the interrupt task is guaranteed, and the resources of the processor due to the scheduling between tasks are also avoided.

3 COMPRESSED RADAR TRACK SCHEDULING ALGORITHM

The scheduling algorithm is applied to the radar track compression algorithm. Experiments show that the data can be received and compressed in real time. The main task of the algorithm is five parts. Respectively receive data in real time, data batch, threat judgment, data compression and data upload. Through the task decomposition, receive interrupts to achieve real-time data reception and balanced execution of tasks.

3.1 System Structure

Real-time data is uploaded from the radar to the processing terminal according to the RS232 protocol. Since the appearance of the target is uncertain and the radar detects that the target needs to be uploaded in real time, otherwise the data will be lost, so the data will be received.

Set to interrupt reception and set the highest priority. And in the cyclic task enable receive interrupt (NVIC_EnableIRQ()).

The terminal receives radar data in bytes. The target track ultimately consists of a series of fixed-form traces, each of which represents the motion parameters of the target at the corresponding time [9]. The first thing you need to do is divide the received bytes into standard format radar traces. Radar track data compression is performed using feature points. Before the compression, it is necessary to acquire the points of a certain period of time of the batch of the target, and batch the received point data according to the target batch number in the fixed data format.

The interval between radar upload bytes is the same. Framing of data can be completed within this interruption interval. Due to the relatively long time required for data batching, placing batches in receiving interrupts will result in the omission of target receiving points. Batching data in cyclic tasks. Set the flag. Query the flags in the cyclic task and batch the data.

Threat judgment is the screening of different batches of targets. By querying the new target flag, the threat assessment function is used to calculate the new target threat degree. For the radar to obtain the track, it needs a certain period. In order to balance the operation, a recursive algorithm is used to calculate the threat degree of the target. Query the time period flag to perform data compression tasks. When the data is compressed, the compressed data is uploaded through the serial port. A simple flow chart for a compressed radar track scheduling algorithm is shown in Fig. 3.

Fig. 3. Simple flow chart of compressed radar track scheduling algorithm

3.2 System Implementation

The system is implemented according to the designed system structure. The main functions, subfunctions, and flag bits of the system are shown below. The main program contains two while loops, querying the frame flag bit in the first while loop, completing batches of points. Query the new target flag to complete the threat assessment of the target. After the time period, jump out of the first while loop. The trajectory is compressed based on linear prediction. The feature points of the target track are uploaded and the related flag bits are cleared to complete the target track compression and upload for one cycle.

```

Int main(void)
{SystemInit(); //System and variable initialization
UART1_Init(); // Serial port initialization and setting
receive interrupt
if(flag1==1) // Received radar track
BD_date[Date_qdbnew][j++]=ch; // Framing the points
... }
While (1)
{while (1)
{if(flag2==1) //Data receive and frame flag
For (i = 0; i < 20; i++) // batches the framing points
{if(get_same_pi[i][0][5]==BD_date[t][5]) }
If(flag4==0) //new track is a new target
If(flag4==0) //new track is a new target
{wei[n]=DATEHANDLE(n); //Calculate the threat of the
new target}
MAO(wei[n]); //Bubble sorts the target's threat
}
}
}
    
```

```
If(timer0_counter>=6000 && n!=0 )
Break; //out of cycle at 1 minute of set time period}
A0=fana0(0,0, julix, juli); // Call function calculation
coefficient (batch number, starting point of fitting)
A1=fana0(0,0, julix, juli);
For(i=3;i<dian_num;i++) //Finding feature points based
on linear prediction
For(j=0;j<76;j++)
{ UART3_SendByte(dispose_date[j]);//Upload feature
points
Dispose_date[j]=0x00;}
Timer0_counter=0; // Clear the relevant flag bit
...} }
```

3.3 Effect of Operation

The designed scheduler was put into operation in LPC1768. Experiments showed that the data uploaded by radar can be processed in real time. And it can screen out high threat targets through threat judgment. Can effectively compress the target track and upload data effectively. Due to the large amount of target track data, it is easy to lose the target data and cannot effectively screen high-threat targets. Through the interrupt call, the program execution time is reasonably allocated. By decomposing complex tasks, framing of radar traces can be completed within the interval of received bytes. Reduced system delays and improved processor efficiency.

4 CONCLUSION

With the increasing complexity of tasks handled by embedded processors, the ability of processors to efficiently schedule tasks for execution has become a key factor in determining the efficient use of processors. Through the decomposition of the complex tasks into simple tasks proposed in this paper, simple tasks can be completed within the interval of the processor performing other tasks, reducing the system delay, thereby improving the efficiency of the processor. At the same time, the

reasonable scheduling of interrupt tasks and cyclic tasks completes the reasonable time allocation of the processor, avoids the system delay caused by the waiting tasks, and then increases the speed of executing the tasks of the processor.

REFERENCES

1. Zhou Lei. Research and implementation of event-driven embedded system scheduling kernel [D]. *Jilin University: Jilin University*, 2015.
2. Sun Juan. Design and implementation of task scheduling algorithm based on service quality [D]. *Dalian University of Technology: Dalian University of Technology*, 2016.
3. Jiang Jianchun. Research on key issues of heterogeneous multi-core embedded software [D]. *Chongqing University: Chongqing University*, 2012.
4. Cheng Yi, Zhao Hongwei, Long Manli, Li Yucui,. Improvement of the earliest deadline scheduling algorithm [J]. *Journal of Jilin University (Engineering Science)*, 2013, (5).
5. Jiang Lei,. Research on preemptive scheduling algorithm and preemption threshold scheduling algorithm [J]. *Modern Computer*, 2013, (12).
6. Li Zhe, Mu Dejun, Guo Lantian, Huang Xingli, Li Liutao. Research on Mixed Scheduling Mechanism of Embedded Multiprocessor Systems[J]. *Journal of Northwestern Polytechnical University*, 2015, (1).
7. Xiao Hong, Zhou Puxiong,. Embedded Multi-core System Software Design and Development [J]. *Modern Computer (Professional Edition)*, 2008, (5).
8. Wang Liang. Research on large-scale grid platform modeling and scheduling algorithm [D]. *Xidian University: Xidian University*, 2005.
9. Song Yan, Liu Limin, Han Zhuangzhi,. Research on Radar Track Data Compression Method[J]. *Electrooptics and Control*, 2017, (4).