International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 04
February 2018

A Review of Embedded System Memory Management

P.Suneel Kumar & Dr.N.Kumarappan
1. M.Tech., Ph.D Scholar, Detp. Of ECE, Annamalai University, Annamalai Nagar, Chidambaram, Tamil Nadu, India.
2. M.Tech., Ph.D, Professor, Detp. Of ECE, Annamalai Nagar, Chidambaram, Tamil Nadu, India.

Abstract— *This paper considers the fundamental prerequisites of embedded operating system memory management and the key issues in memory management, by means of dissect the points of interest and impediments of various memory management calculation, as indicated by the highlights of the embedded system, for example, the mount of the memory is few, and has no MMU, and has continuous necessities generally, Then proposed a strategy that can appropriate and reuse the memory quickly, and the operation time to the memory is sure, so It's ready to address the issues of embedded operating system's constant execution necessities viably, in addition utilize this way to deal with oversee memory can keep away from intemperate memory sections.*

Keywords-component; memory management; buddy algorithm; embedded operating system, Embedded OS

I. INTRODUCTION

The Embedded System configuration has turned into a noteworthy field of current computer application, and the course of present day computer advancement. Embedded system can use in numerous spaces, for example, robotization field, vehicle field, Portable hardware, aviation, weapon types of gear and also different perspectives throughout everyday life, so the Embedded system has a decent prospect of use. With a specific end goal to adjust expanding assorted variety and multifaceted nature of the application, utilizing Embedded operating systems in the embedded systems has turned into a bearing for the future improvement of embedded systems. Since utilizing the embedded ongoing operating system (RTOS) can be all the more soundly and productively to convey the CPU asset and different assets, streamline the outline of use programming, abbreviate the season of system improvement, guarantee the constant execution and unwavering quality of the system.

The exploration of embedded operating system memory management procedure has turned into a key area in embedded operating system, in light of the fact that the Embedded operating systems and the broadly useful operating system has a ton of contrasts. Most embedded system has the constant prerequisites, with a specific end goal to meet that point, the dispersion and reuse of system memory must be quick and solid. So it is difficult to utilize confuse memory assignment methodologies in the embedded operating system, yet ought to be as basic and quick as could be allowed. Additionally, numerous embedded processors don't have virtual memory management units(MMU), without the virtual memory management unit, the entrance of memory is specifically, and the addresses the undertaking need to get to are real physical address. There is likewise no address security for the operating system when without MMU, all running undertaking Sharing a similar memory space. The errand keeping in mind the end goal to execute must get sufficient ceaseless memory space and load all code and information to that memory before it run. Amid the running of the undertaking , if the assignment absence of memory , it can't get more by trading the substance in memory to the plate, on the grounds that the system does not bolster this procedure. For a system that has virtual memory management strategy, for example, Linux , there is no compelling reason to designate all memory for the errand in the meantime, likewise the memory disseminated for the assignment does

not should be a piece of constant physical space, what the system require is to guarantee the virtual memory address consistent ,that is will be alright. In the system without virtual memory management procedure, the client application and the portion has a similar space [2], when build up the program, the software engineer must guarantee one assignment does not get to the next errand's space, generally the system will breakdown.

II. MEMORY MANAGEMENT APPROACHES

There are two sorts of Memory management methodologies, one is static memory designation procedure, and another is dynamic memory portion technique. In the event that utilization Static assignment methodology the extent of the memory space was resolved when accumulate the system. While the system introduce, a settled measure of memory required to store the question and information was apportioned. The undertakings utilize the memory assigned for them when they run, notwithstanding when the errands was finished, the memory that was involved by the undertaking won't be discharged. So static portion procedure will squander a considerable measure of memory, in light of the fact that the memory assigned for the errand must as indicated by the most exceedingly awful condition, actually, the undertaking will utilize just a little piece of the memory, and a lot of the memory was squandered. Additionally, adding a few capacities to the system that utilizing static memory management methodology isn't generally helpful, so that, redesigning the system turns out to be extremely troublesome.

he dynamic memory distribution procedure will get the memory just when it's vital amid the errand running, and discharge the memory while the undertaking was finished [3]. All memory management operations are given by the operating system's memory management module.

Both of Static and dynamic distribution procedure has their own particular focal points and impediments. The trial of the program that utilization static memory distribution methodology will be simpler, and the execution will be more effective, yet takes more memory. Furthermore, the dynamic allotment methodology is more adaptable, the main hindrance is that it ought to apportion some memory space to store the management data, and possesses extra system assets and inclined to bring forth more memory pieces, more finished the season of dynamic memory dissemination and reuse is indeterminate.

III. THE DESIGN OF EMBEDDED SYSTEM MEMORY MANAGEMENT

Contrasted with the work area operating system, embedded operating system's memory management has its own highlights. Initially, the embedded system was bound by a few variables, for example, the costs, the size et cetera, the memory in the system generally little, so the utilization of memory must utilize effectively. Furthermore, embedded system as a rule have the necessities of continuous, consequently, memory portion and reuse must be expeditious, in the meantime embedded system equipment from time to time redesign after the item was issued, so don't require consider the memory grow. We should utilize distinctive memory management procedure in Embedded systems as per the particular application.

A. Buddy algorithm

Buddy algorithm is a sort of memory management strategy, this procedure through merger and split the memory square to deal with the memory. The system merger two little squares to shape an extensive piece that the memory address is constant so it can address the issue of errand which must get a vast memory piece. What's more, part the extensive square when it's essential at that point give a piece to assignment that request that size memory, and the remained was leaved for others, so the system can spare the memory. The primal buddy algorithm have a few issues, since when one assignment discharge a piece then the system endeavor to combine it with others instantly, and

if simply after the blending, some undertaking demand that size of square once more, however the system does not have that size of memory at present, so the system should part a substantial piece once more, if the consolidating and part activity happen too every now and again, the buddy algorithm may cause stun, in the most pessimistic scenario the system's execution will drop. Obviously, we can enhance the buddy algorithm to enhance the system's execution, one enhanced buddy algorithm is called sluggish buddy algorithm, the perfect of the lethargic buddy algorithm is to defer the union time when the errand discharge the piece, and let the blending activity happen just when it's important.

B. The design of buddy algorithm control structure

In this paper, we outlined a memory management system utilizing the paired accomplice algorithm. The system separated the memory space with the exception of the space as of now apportioned for the system bit and the stack to N allotments. The span of each square is 2nKB, the n was the parcel number, and connection the pieces inside the segment to shape a bidirectional chain[3][4]. The information structure for the parcel control MEMEY_PARTITION_CB and the memory square MEMERY_ BLOCK_CB are as per the following :

```
Typedef struct
  memery_partition_cb{
  MEMERY_BLOCK_CB
  *first_block_pt;;
        /*the first block in the
  partition*/ UINT32
  *start_addr_pt;
        /* partition start address*/
  UINT32 *end_addr_pt; /*partition end
  address*/ UINT16 total_num; /* block
  number */
  UINT16 free_num; /*free block
  number*/ UINT16 block_size; /*the
  size of the block*/
  }MEMEY_PARTITION
_CB Typedef struct
memery_block_cb{
        MEMEY_BLOCK_CB
*block_pre_pt; /*point to the
        previous block*/
        MEMEY_BLOCK_CB
        *block_next_pt;
    /*point to the next block*/
UINT16 size; /* block primitive size, equal to
        block _size*/
UINT16 current_size; /*block current
size*/ UINT32 *start_addr_pt; /* block
start address */ UINT32 *end_addr_pt;
/* block end address */ UINT8 state;
/*the state of the block */ UINT16
task_id; /* task id that is using the
block*/ }MEMEY_BLOCK_CB
```

The buddy algorithm memory management strategy is easy to implement, and the efficient of memory allocation and recycle is very high, can satisfy the embedded system‘s requirement of real-time. The relationship between partitions and blocks within the partition was show as figure 1.

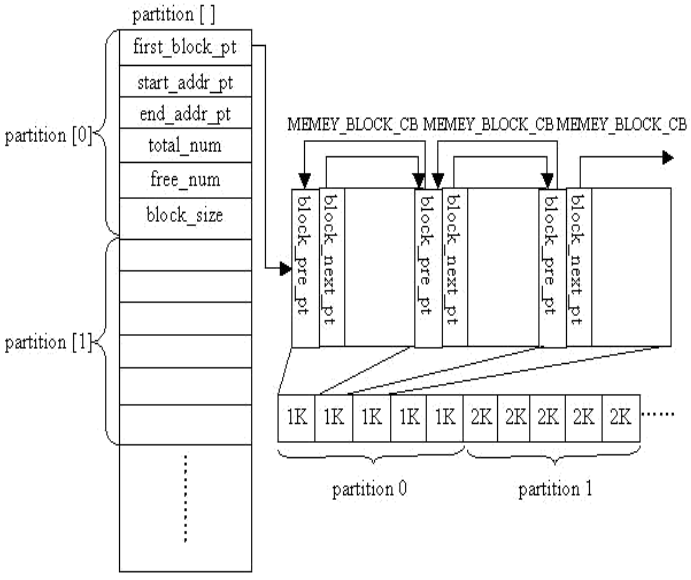

Figure 1. Relationship between partition and block

At the point when the undertaking approach the system for memory, the system will compute the measure of memory that the assignment demand to figure out which segment the littlest memory obstruct that can address the issue of the

errand will be in, if that parcel have free pieces, at that point the system will get a free square for that assignment, generally the system will part a substantial piece in the following allotment, one section for the undertaking and the other part leave for different errands. Expecting the assignment require 1KB memory space, the system will test segment 0 to begin with ,in light of the fact that the square size of parcel is 1KB,if the segment 0 has pieces accessible ,at that point the system take a piece for the errand. in the event that parcel 0 does not have squares accessible, at that point the system will seek segment 1,if segment 1 has pieces accessible at that point get one piece and split the piece into two sections, so each of them is 1KB,the initial segment put to segment 0's piece chain, and the second part designate for the assignment. On the off chance that the segment 1 still have no free pieces, at that point test parcel 2,if it can get free square in segment 2,the square size will be 4KB,then first split the 4KB piece into two sections ,each of them was 2KB,the initial segment 2KB piece put to the segment 1 piece chain, and keep on splitting the second 2KB square into two sections, the initial 1KB piece put to segment 0 piece chain, and the second part distribute to the undertaking demand for the memory. In the event that parcel 2 additionally don't have free pieces, at that point rehash the means like some time recently, until there is no free squares expansive than the undertaking request in all segments, at that point suspend the errand.

When release a block, the system will search the block chain it current belong to according to current size , if the chain have buddy block, then merge the two buddy blocks to form a large block ,and to the next partition to find buddy of the formed block again, if there has buddy too then merge the two buddy blocks again to form a even large block, repeat these operations until there has no buddy block. The two block to become buddy blocks, they must satisfy three conditions simultaneously: first, they must have the same size. Second, the physical address of the two blocks must adjacent. Third, the two block must split from the same partition.

C. The design of the partition bitmap

With a specific end goal to discover the square fulfill the necessities quickly, the system utilize a bitmap to demonstrate the condition of the parcel's piece chain. There are two factors: MemGrp and MemTbl[8], MemGrp is a Byte, has 8 bits, and MemTbl[8] is an exhibit, has 8 components, every component is a Byte, so the cluster MemTbl[8] has 64 bits, each piece in it shows the condition of a parcel's square chain. On the off chance that the chain has free squares then the bit Associate to it was set 1, else it was set 0. Keeping in mind the end goal to enhance the hunt speed, we plan a pursuit table in the system, when to discover the square chain fulfill the need, just needs utilize MemGrp and compare component in the cluster MemTbl[ ] to look into the inquiry table SearchBit[ ]. SearchBit[ ]is a const exhibit, its estimation was traversed ascertain. The exhibit SearchBit[ ] was appear as take after:

```
UINT8 const SearchBit[] = {
0, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
7, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0
}
```

D. Search the block chain

he operation to look into the piece affixes is to locate the main square chain that has free squares and the piece estimate fulfill the requirements. In the system it has numerous piece chains ,if cross all chains one by one then it will require a ton of investment, in the interim the time is

questionable, this won't permitted in the constant system. So we utilize a bitmap said in 3.3 to tackle the issue. the connection amongst MemGrp and MemTbl[ ] appear as figure 2.

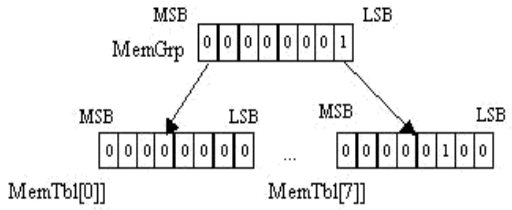

Figure 2. bitmap

The bitmap is similar to a 8 branches tree, every hub has 8 leaves, the root is MemGrp ,its leaves are MemTbl[0] ~ MemTbl[7], and the components MemTbl[0] ~ MemTbl[7] likewise has 8 bits in each of them as their leaves, each piece partner to a square chain. When we need to discover the chain we require , do operations as take after:

(1) according to the memory estimate the undertaking solicitation can get the segment number, utilize PN to indicate it.

$$PN=\left\lceil \log_2^{size/1024} \right\rceil$$

(2) use PN to get which block chain group it might in, use CG to denote the chain group, CG= $PN / 8 = PN>>$ 3, and the bit position in the group use GP to denote, GP=PN&0x07

(3)N=MemTbl[PN]&(~((1<<GP)-1)) so it can mask the bits in MemTbl[PN] lower than GP, if N≠0, then use SearchBit[ ]to get the position of the lowest bit in N, so the block chain number R= CG<<3+SearchBit[N],and R is the result we want.

(4)If N=0 then,M= MemGrp&(~((1<<CG) - 1)), use this method can mask the bits in MemGrp lower than the bit position CG the task request. If M=0 then there is no block chain available, the system will return an error message and suspend the task ask for the memory. If M≠0, then B=SearchBit[M], R=B<<3+SearchBit[MemTbl[B]], and the R is the result we want.

If allocate a block and the chain become empty, then the system must modify the bit associate to it, meanwhile if group become empty also, then the correspond bit in MemGrp should be modified too.

IV. Conclusion

In this paper we talked about a sort of memory management technique in view of twofold accomplice algorithm. what's more, utilizing the bitmap to demonstrate the piece chains, makes the operations of memory assignment more effective and quick, the execution is superior to navigate the square chains, and the season of the operation to the memory is sure. At the point when the system do the operation of designating memory, it can handicap intrudes, in light of the fact that the season of it is short. Utilizing the technique for empower/incapacitate hinder can maintain a strategic distance from the need reversal issues

V. References

[1] Preeti Ranjan Panda and Nikil D. Dutt,Memory Architectures for Embedded Systems-On-Chip[J], Springer-Verlag Berlin Heidelberg 2002

[2] Liu Dong-dong. Research and Improvement of VxWorks Memory Management Mechanism[J]. Science Technology and Engineering. 2007(6) .

[3] Zeng Fei-yi , Sang nan , Xiong Gong-ze. Memory management research of embedded system[J]. Microcontrollers & Embedded Systems. 2005(1) .

[4] Huang Xian-ying , Wang Yue , Chen Yuan. Memory management strategy in embedded real-time system[J]. computer engineering and design.2004(10)

[5] Zhao Yue-hua, Cai Gui-xian, Huang Wei-ju. Design and implementation of embedded and secure memory management[J].Computer Engineering and Design.2006(16) .