PROCEEDINGS

OF THE INTERNATIONAL SYMPOSIUM'90 ON COMPUTER APPLICATIONS IN INDUSTRIAL AUTOMATION

计算机在工业自动化中的应用中外研讨会论文集

中国有色金属学会计算机应用学委会

JUNE 1990 BEIJING

PROCEEDINGS

OF THE INTERNATIONAL SYMPOSIUM'90

ON COMPUTER APPLICATIONS IN INDUSTRIAL AUTOMATION

计算机在工业自动化中的应用

中外研讨会论文集

（下）

中国有色金属学会计算机应用学委会

1990年6月　北京

目 录 （下册）

CONTENTS (Vol. 2)

SOS-A SMALL OPERATING SYSTEM FOR EMBEDDED SYSTEMS

Jerry E. Sayers
East Tennessee State University
Johnson City, Tennessee 37614

Abstract

A multitasking operating system for IBM PC or compatible has been developed to provide students with an educational opportunity for hands on experience with concurrent programming concepts. The operating system includes featues which permit users to observe performance a function of the I/O versus compute bound processes. Although other such educational operating system exits, this operating system is limited primarily to process management. The small size of the operating system makes it easy to comprehend in a single term course. It is, however, a powerful real time executive for applications such as small, embedded system.

Introduction

There are many definitions for an operating system. There are many operating systems in use and they accommodate a wide range of applications. However, an operating system may be defined simply as a set of software designed to assist a user of a system in utilizing the computer hardware and its resources. The resources include programs, data, and peripherals.

Despite the variety found in computing hardware and their operating systems, there are many common elements found in all operating systems. The intent of this paper is to give an overview of the basic clements of an operating system and a discussion of the evolution of operating systems development. The driving forces behind that development will be preseted. We shall examine the basic types of operating systems and typical applications best suited for them, develop a logical design and give an implemantation of a small operating system (SOS) which will run on IBM/PC/XT/AT and 100% compatibles.

SOS is a simple, though functional, fixed priority, preemptive operating system (more about that later) suitable for dedicated applications such as might be found in any application requiring real time response to a few multi tasked processes. The application used in this paper is that of data collection platform with minimal data management capabilities. The SOS kenel (the basic set of operating system primaritive processes) should be adaptable to a wide variety of similar applications. However, the main purpose of this paper is to provide an educational experience with a multitasking operating system augment by a usabl, hands on application. The basic elements found in SOS are also found in all other multitasking systems and should provide a good beginning for students of operating systems.

The Evolution of Operating Systems

Operating System Concepts

Types of Operating Systems

Description of SOS

The logical design of Small Operating System (SOS) will be developed, and the data structure to support an implementation of SOS will be presented. SOS is a fixed priority, preemtive, real time, multitasking system. It consists of a kenel and application tasks. The kenel is the nucleus of an operating system and SOS's kenel will contain system processes, an interrupt subsystem, and support functions such as system start up, system shutdown etc... It would be ideal for very small embedded systems. The system will provide the frame work for a sixteen process (or task) system. The Scheduler is allocated to Task 0, the highest priority task in the system. We shall develop a system with, minimally, tow tasks, an interrupt subsystem and some support function that make up the kenel. The tasks are the Scheduler and the Work Up task. The inerrupt service routines to be written are the Timer handler. A common front of timer interrupt and voluntary suspends will be a part of the interrupt subsystem. The duties of the support procedures will include the following:

.save and load new interrupt vectors

.restore old interrupt vectors

.initialize timer, printer, com port, etc...

.set, reset interrupt masks

.initialize PCBs

.provide direct memory video display

Some application processes will be written to demonstrate the use of SOS. These will be 1) a keyboard input manager, 2) aserial communications manager, and all remaining task slots will be filled with CPU intensive tasks to demonstrate the multitasking ability of SOS. A more detailed discussion of the processes, the interrupt subsystem, and the support functions will be given later. First, let us re explore the concept of a process.

Processes in SOS

The Interrupt Handler/Interrupt Subsystem

Interprocess Communication

Logical Model Diagrams

From the descriptions given above, two diagrams were designed. Figure 2. is a data flow diagram (DFD) which shows graghically how information flows in the SOS environment. In particular, note that the DFD is partitioned into the interrupt subsystem, the multitasking subsystem, and the SOS/MS-DOS domains. Figure 3. is an entity relationship diagram (ERD) which shows the relationship between different objects in the SOS system. These type of diagrams aid in the design of a software system, and the work done developing them is helpful in identifying potential problems before too much effort is committed to the implementation phase of the software life cycle.

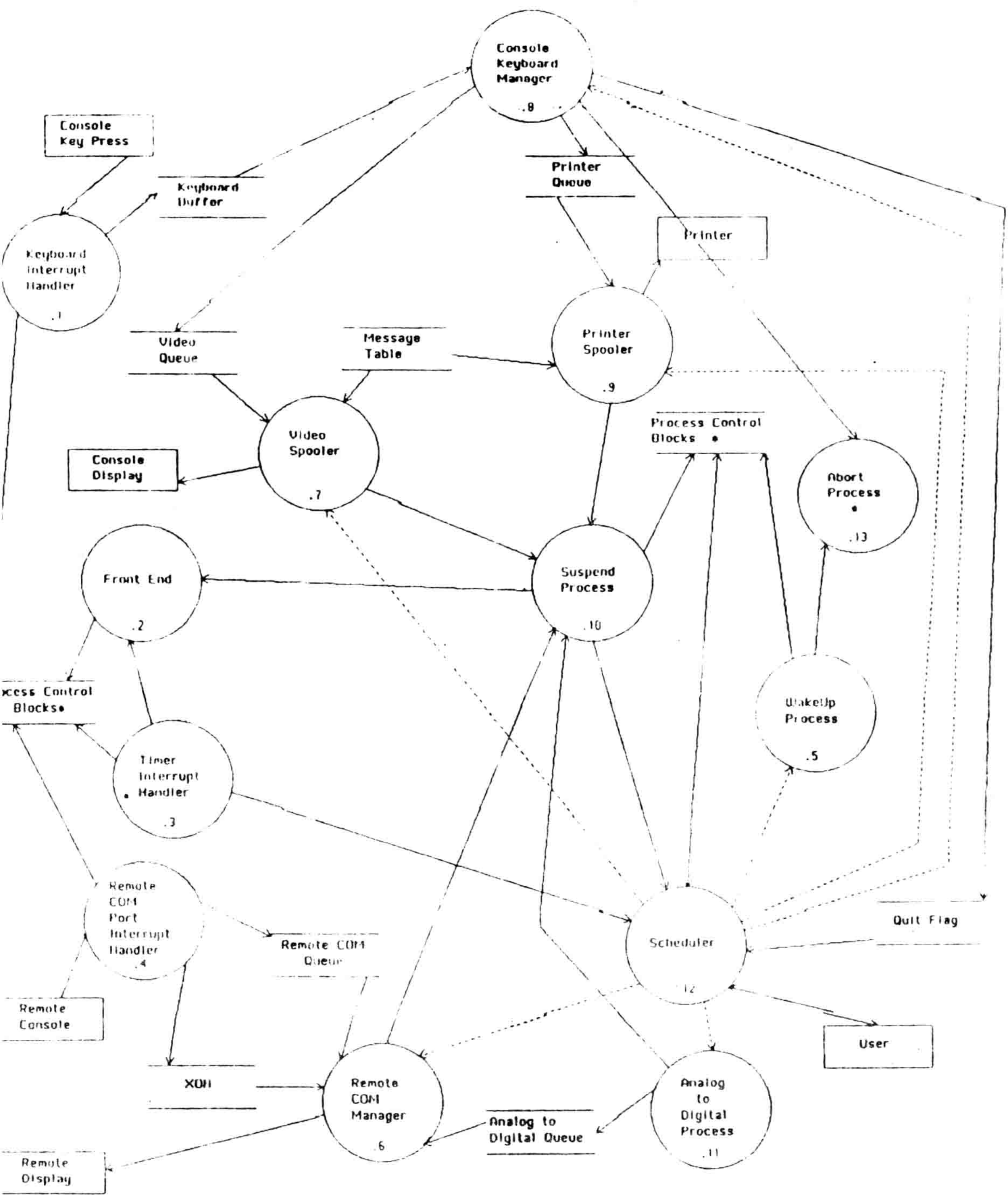
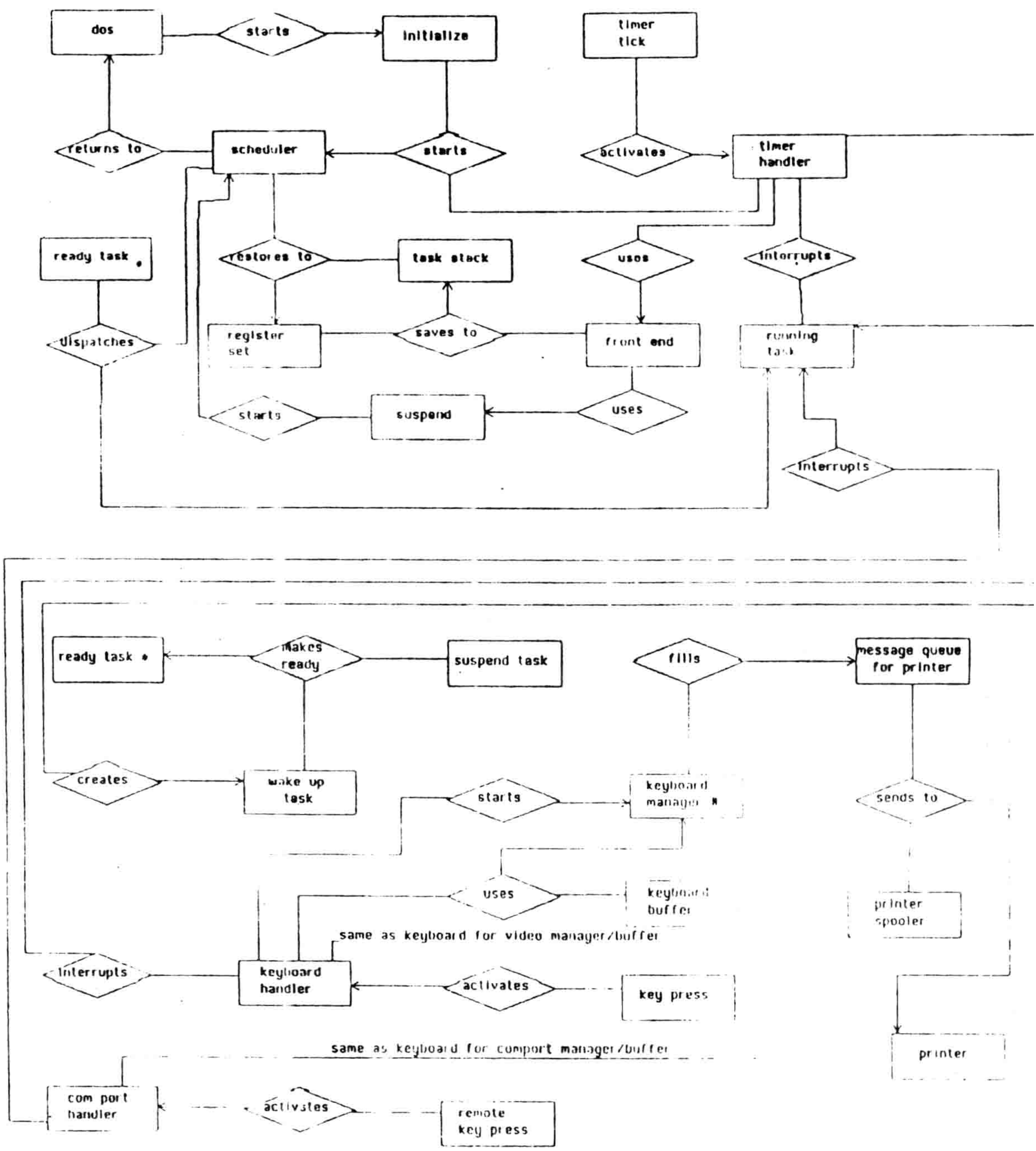


Figure 2. Data and Control in SOS



*Duplicated on Diagram

Figure 3 Entity Relationship Diagram for SOS

BIOS/BDOS Problems

Some Other Operating Systems

Basic Design Considerations

The SOS Kenel

Application Processes in SOS

Cyclical Versus Aborted Processes

Queue Management Processes

Special Features of SOS

Two special features were added to SOS for debugging and demonstration of SOS's multitasking capabilities. One feature was fill all unused process slots with a compute bound process and to permit the user to modify the amount of time these processes needed the CPU. The other feature was to have a real time display of the status of each process.

SOS dynamically displays the status of each process as shown in figure 5. This feature requires a lot of overhead, but provides a dynamic view into the states of the operating system processes. The performence loss is acceptable becauce of the educational benefit derived from this feature. An ancillary benefit of the system status display is its debugging potential. It is easy to verify that transitions of a process occur as excepted by viewing the status display. Other features designed to enhance the educational value of the operating system were implemented in some of the user processes. For example, the compute bound processes all are infinite loops which consume CPU cycles for a variable amount of time, and then sleep for a fixed amount of time. The times are directly related to the PID value times a multiplicative factor; thus, the lower the priority, the longer the loop between SUSPEND calls. By pressing the keypad "+" or keypad "-" keys, the multiplicative factor to an amount that causes the low priority processes to be suspended most of the time. This was done purely for educational purposes. The code used to create several of these processes was

```
PID9      EQU      12               ;Define PIDs for
PIDA      EQU      14               ; processes 9 & A

          MAKE_TASK 9               ;Generates a complete PROC.
          MAKE_TASK A               ; -place after an ENDP statement.
```

Place in Start Up code to "create" the process on the READY list

```
           MOV AX,1                 ;Status bit for READY
           MOV STATUS[PID9],AX      ; -set for Process 9
           MOV STATUS[PIDA],AX      ; -set for Process A
```

Listing 5 is the macro for creating the compute bound processes.

Small Operating System Status

Task	Task
0: -NUL-	1:Aborted
2:Aborted	3:Suspended
4:Suspended	5:Suspended
6:Suspended	7:Running
8:Suspended	9:Ready
A:Ready	B:Ready
C:Suspended	D:Ready
E:Suspended	F:Ready

Figure 5. System Status Real Time Display

```
MAKE_TASK     MACRO     N
TASK&N&:      PROC      FAR
TSTART&N&:    ST
              MOV       AX,RUNNING        ;Get PID&N&
              MOV       AH,MULTIPLIER     ;Put multiplier in AH
              MUL       AH
              MOV       CX,AX
TLOOP&N&:     PUSH      CX
              MOV       CX,4000h          ;Set count (*MULTIPLIER*PID)
TWAIT&N&:     LOOP      TWAIT&N&          ;COMPUTE BOUND !
              POP       CX
              LOOP      TLOOP&N&
              MOV       AX,RUNNING        ;Get PID&N&
              MOV       AH,55             ;COUNTS FOR 1 second
              MUL       AH                ;Prepare to wait
              PUSHF                       ;PID
              CLI                         ;seconds
              CALL      SUSPEND
              JMP       TSTART&N&         ;Restart task
TASK&N&:      ENDP
              ENDM
```

Listing 5. Macro to Create Compute Bound Processes.

A Sample Apllication

Conclusion

We have looked briefly at the evolution of operating systems. We have had an overview of the basic components of an operating system as well as a discussion of several flavors of operating systems and the application best suited for them. The purpose of this paper was to lay the ground work for the design, develoment, and implementation of small, fixed priority, peemptive, real time operating system named SOS (for Small Operating System). Next the paper presented a design for SOS. SOS is written in 8086 assembly language. Although it was undertaken as an educational project to learn about multiprocessing, the resulting product is a very powerful and useful operating system when used within the range of applications for which it was intended.

The reader is referred to the Selected References for further reading. The book by Davis is a user view of operating systems. The book by Maekava et al covers advanced topics like performance measurement and queuing theory. The other books fall into two catigories; generic concepts with a hands on operating system developed in the book. Of special interest for this paper is the text by Milenkovic, in which a PC based Real Time operating system is presentde. The text by Tanenbaum develops a multiprocessing system Minix which is similar to Unix. Likewise, the text by Comer develops another Unix work alike operating system named Xinu. The others are more generic or are based on a commercial operating system (e.g., Yuen uses the IBM mainframe as his example machine) and represents many of the "classical" operating system books available today. Hopefully, this list will provide you with a good starting place to learn more about operating systems.

SOS——一个用于嵌入系统的小型操作系统

美国东田纳西州立大学　J．E．塞叶斯

摘　要

一种用于ＩＢＭ　ＰＣ或其兼容机的多任务操作系统已经开发出来，它向学生提供学习并行程序概念时亲手实践的环境。该操作系统的特点是允许用户观察Ｉ／Ｏ功能相对于有关计算过程的运行性能。虽然已有其他同类教学用的操作系统，但这些操作系统主要限于进程管理。由于该操作系统的规模小，使它容易在一个学期的课程中来领会。此外它也是一个强有力的实时执行机构，可用于小的嵌入系统中。

标　题

引言，操作系统的演变，操作系统的演变，操作系统的概念，操作系统的类型，ＳＯＳ中的进程，中断处理器／中断子系统，进程通信，逻辑模型图，ＳＯＳ的早期发展，ＢＩＯＳ／ＢＤＯＳ问题，某些其他操作系统，基本设计考虑，ＳＯＳ核心，ＳＯＳ的应用进程，周期性地撤销进程，排队管理进程，ＳＯＳ的特征，采样应用，结论。

结　论

本文简要回顾了操作系统的演变，概述了操作系统的基本组成部分，讨论了操作系统的一些特点及其最适合的应用。本文的目的是为设计、开发和实现一个有固定优先级的小型实时操作系统ＳＯＳ（小型操作系统的简称）打下基础。其次，文中介绍了ＳＯＳ的一种设计方法，它是用８０８６汇编语言编写的。尽管它是作为一种学习多任务处理的教学目的而设计的，但其结果的产物则已形成一个在予期应用范围内的强有力的实用的操作系统。