STRUCTURED PERFORMANCE EVALUATION

W.C. LYNCH

Rapporteurs: Dr. S. Tsur
Dr. I. Mitrani
Dr. J.H. Hine

Abstract

The structured programming technique of successive refinement can be extended to include a hierarchical description of performance quantities and measures. This technique can be used at design time for estimating performance and can be used after implementation as a framework for performance measurement. As an example, the technique is applied to understand the measured performance of the CHI/OS system.

This talk is devoted to the description of a quantitative analog of structured programming. A top down technique together with typical values for system variables will be examined. The aim is to produce a priori estimates of the the behavior of large systems, particularly operating systems. It should be mentioned that this is *not* what we set out to do. Rather, the technique forced itself upon us as the only way of being able to make sense out of what otherwise was a large number of unstructured measurements.

Some of the things that can go wrong in operating systems will be shown. The assertion is that the problems we encountered have been very high level system design problems, in particular, feedback control problems. We have not observed problems as a result of the incapability of the hardware to carry the load.

The accompanying figures and diagrams were produced by Mr. E. Klein as a part of his M.Sc. thesis.

Introduction

What is meant by structured performance evaluation? This is a combination of structured programming and an examination of certain quantitative measurements. In particular within an operating system, the technique consists of two parts:

1. A qualitative description of the system in a structured top down manner. This description consists of a small number of modules and their interconnections. Each of the modules in turn is described by a standard hierarchical explication such that the combination of the I/O relations of each of the component boxes must correspond to the I/O relations of the enclosing generalised block.

2. A quantitative performance evaluation. In this case, the performance measure used was the *module call rate*, i.e., the rate at which each module exercises its underlings. Although this may seem a very simplified analysis, its derivation required a painstaking process, and it turned out that the results were more relevant than was initially thought. The assertion is that it is impossible to estimate the I/O rates of the various modules without going through much of the analysis.

In general, there are three factors that modify the flow rate through a module.

1. **Blocking Factors** Very often, a module, upon call, sets aside a larger block for buffering purposes. The result is that fewer requests come out of that module than enter it, i.e., a *reduction* of the traffic through the module.

2. **Paging and Cacheing:** These effects exist in particular in the more sophisticated operating systems, often in disguises unknown to the designer. For example, disk arms on disk packs (luckily) do not move often as a result of locality phenomena in the disk references. The net result of these effects is to reduce the traffic through a module.

3. **Index Accesses:** The effect of accessing indices for each I/O operation. Often more than one index is accessed for each I/O operation. These effects tend to *increase* the traffic rate through a module.

A consequence of these factors is that a naive approach towards measuring system performance is often meaningless. A simple measurement of the request rate through some I/O channel, for example, would be meaningless since we cannot distinguish between the various types of requests.

The example that will be used throughout is taken from the CHI/OS operating system. The first question of interest is: what is the load imposed on the operating system? A system load summary is given in Figure 1. From the summary, we can see how many times the operating system is accessed per unit time, i.e., how many ER's were executed per unit time. This is the *request rate.* The unit time chosen is *User Second.* Very early in the project, it was learned that a unit time of real seconds is unsuitable, primarily because the results depend heavily on the extent of the operating system execution time. In the case of CHI/OS, which runs on a UNIVAC 1100, the distinction is simple: the operating system executes entirely in executive mode and the user entirely in user mode. These modes are displayed in the computers' PSW. In this case, User Code includes the compilers, any execution of a library and the execution of the linking loaders. It does not include the access methods. When the figures from this system are compared with those of other systems, one has to be careful since both the definition of user mode and the time unit used may be different. We can see from Figure 1 that the operating system is called approximately 300 times per 10^6 user instructions. Assuming that each ER is on the order of 3000 instructions, we see that, on the average, one operating system instruction is executed for each user instruction.

Another interesting number is that approximately 89 print lines are generated for each 10^6 user instructions. This is rather high, 60 is more likely. This number appears to be dependent on the systems' turnaround time. When the turnaround time is long, users tend to print more data, which in turn increases the executive overhead, which in turn increases the turnaround time. This is a typical example of a positive feedback control loop. Similar loops exist all over the system and they may be unstable. Such loops, which are the really important factors in an operating system design, are seldom taken into consideration.

The idle time in Figure 1 is caused by two factors:

1. No work in the machine, i.e., temporary low load.

2. No overlap between processing and I/O.

Figure 2 shows the idle time distribution as a function of the number of users in the system. It is always possible to fit two users into core. Thus, from these results, it was concluded that if there are one or two users in core, the system is idle because of insufficient load. We arbitrarily say that if there are two or more users in core, the system is idle because of I/O wait. Figure 2 shows that in CHI/OS almost all idle time is due to a lack of load, with only 5% idle time when two or more users are in core.

Figure 3 shows the system decomposition on the highest level. The three boxes on the top line are the major system drivers. The user programs are lumped together in a box labelled "USERS", the second driver is the spooling subsystem labelled "SPOOLING", and other miscellaneous drivers are lumped in a box labelled "MISC". The flow rates of requests between the boxes are in requests/user second. On top of the figure, the user/cpu/real time ratios are displayed. Consequently, in this case, the user time has to be multiplied by a factor of ~3 to convert it to real time. Some of the more important boxes are labelled with a number. These numbers will be used for further hierarchical explication. The users make calls on the file system. Those calls for files which reside on magnetic tapes are diverted directly to the device handlers. In this system, the file handler is integrated and all the file requests, whether directed by user or not, are handled in a uniform way. Figure 4 is an explication of the "USERS" box. Of the various categories of calls on the Operating System by users, only a few are important. "ITEM OUTPUT" are the requests for print lines "ITEM INPUT" are the card reading requests plus the input from one line terminals. "FILE I/O" are calls on the file system and "TAPE I/O" is split: those calls on real magnetic tapes are directed towards the device handlers and the calls to the simulated magnetic tapes to the file system. Both ITEM OUTPUT and ITEM OUTPUT call on other modules which are subject to blocking. Consequently the output traffic from FWIP and ER HANDLER is reduced with respect to their input traffic. Figure 5 is an explication of the spooling box in Figure 3. As with the user processes, both the CARD READERS and PRINTERS are subjected to blocking which tend to reduce the traffic flow. The actual number of printed lines is higher and is on the order of 90/user second. The remainder of lines being saved on a tape for off-line printing. Figure 6 is an expansion of the MISC box in Figure 3. As can be seen, the total activity from this box is rather low. This is one of the results of this analysis: the relative importance of the MISC box was not known beforehand and could only be ascertained by using this technique. Figure 7 is an explication of the file system. Two distinct functions are carried out in the file system. One, the FILE I/O is concerned with the promotion of pages through the memory hierarchy (i.e., disk, drum, core memory) for processing. The second function INVENTORY PROCESSES, is responsible for clearing pages from memory and the release of unnecessary core memory. From the figure, it can be seen that the second function imposes a substantial load on the system and, since it is invisible to the user, often the tendency is to forget such overheads. Figure 8 is the expansion of the FILE I/O processes. This component, again, comprises of two major subsystems. The left part is mainly concerned with housekeeping. The right part, which draws most of the load, handles the transfer of files. This part receives block transfer requests; that is, an initial address and the number of words to be transferred. The DATA TRANSFER box converts the block requests into page access requests and, as a result, the flow rate out of this box increases by 20-30%. The main reason for this increase is that often, a number of pages have to be accessed for each block request. The page is located through a description tree. This is the function of the FINDSEC module. The FINDSEC module, in turn, calls the RDDIR module and as a result, the traffic increases by a factor of 3. This reflects the fact that for an average location request, the tree is searched 3 levels deep before the description is found. Since a cacheing mechanism is employed, the actual access rate to the disk is much smaller. It is clear that from a superficial examination of the external I/O rates, the actual internal traffic, which is quite heavy, would not be observed. The inventory side of the filing system is expanded in Figure 9. The modules in it have to do with the cleaning up of various resources and, as before, the traffic is reduced due to chacheing mechanism.

When the system went first into operation, the user/supervisor rate was 1:4. This caused the user load to saturate the system. The measured symptom was that too much CPU time was used and there was no idle time at all. The "10%" rule predicts that 90% of the time is spent in the execution of 10% of the code. When this was checked, contrary to the prediction, the distribution of execution time over code was found to be flat. The next step was to examine the I/O rates. These were found to be in the order of 50-60 accesses/sec, well below the capacity of the drum in question. An additional problem was that even if the CPU loading problem would be resolved, the result would be that the I/O rates would increase by a factor of 3, thus saturating the drum. The next question to be asked was: "What is the cause of the drum accesses?" In order to answer this question, the hierarchical load tree, which at that time did not exist, was painstakingly constructed from the rough, unstructured measurements which existed at that time. Eventually, it was found that in the spooling system (Figure 5), the blocking factor of the TRANSMIT routine was found to be 1:1. This routine was designed as a feedback control routine such that the blocking factor would increase under heavy load and decrease under light load. The measured variable used was taken mistakenly as print lines/real sec, which, as mentioned earlier, is a bad measure. The result was that, as the load rose, it generated more lines per user second. Because of increased executive time, the number of lines per real second *decreased* instead of *increased* as desired and the blocking factor was *decreased* instead of *increased.* This is another typical positive, unstable feedback loop which could not have been detected without the detailed hierarchical analysis as described.

Summary

1. One has to go through this technique in order to obtain the data for any kind of more sophisticated queueing analysis.

2. It is impossible to obtain the various values of the system parameters and other variables of interest in an unstructured way. The only method is to obtain them in an orderly, structured fashion.

3. Once the analysis is performed, often other variables whose significance cannot be ascertained a priori come forward. These variables are often the "missing links" in the complete, comprehensive description of the system.

Figure 1

Chi/OS System Load Summary

10 Aug 76 @ 04:32:42 to 10 Aug 76 @ 17:16:50

Table 1 - Run Summary

Number of Runs	1452
Total User Seconds	13918
User sec / Run	9.59

Table 2 - User Virtual Machines

ERs / User sec	264.4
Tape - Mounts / User sec	.023
Tape - Accesses / User sec	10.0
File System - Assigns / User sed	1.646
File System - Accesses / User sec	27.9
Lines Printed / User sec	88.8
Cards Punched / User sec	1.37
Other Item Writes / User sec	1.33
Cards Read / User sec	20.2
Other Itwem Reads / User sec	23.4
Program Loads / User sec	1.04

Table 3 - CPU Time Distribution

Total User Seconds		13918
Total Supervisor Seconds		15737
Total Idle Seconds		16650
Total	~12.86 hours =	46305

Table 4 - Open Runs and Runs in Core by Real Time

Number of Users	Open (% of Time)	Active (% of Time)
0	22.1	22.1
1	12.6	13.1
2	11.1	23.6
3	11.1	27.1
4	8.5	12.7
5	34.6	1.3
Total	100.0	100.0

Figure 2

Table 5 - Idle Time by Number of Active Users

Number of Users	% of Idle Time	% of Real Time
0	59.6	21.4
1	27.6	9.9
2	9.8	3.5
3	2.6	0.9
4	0.3	0.1
5	0.0	0.0
Total	100.0	35.80

Table 6 - CPU Time Distribution

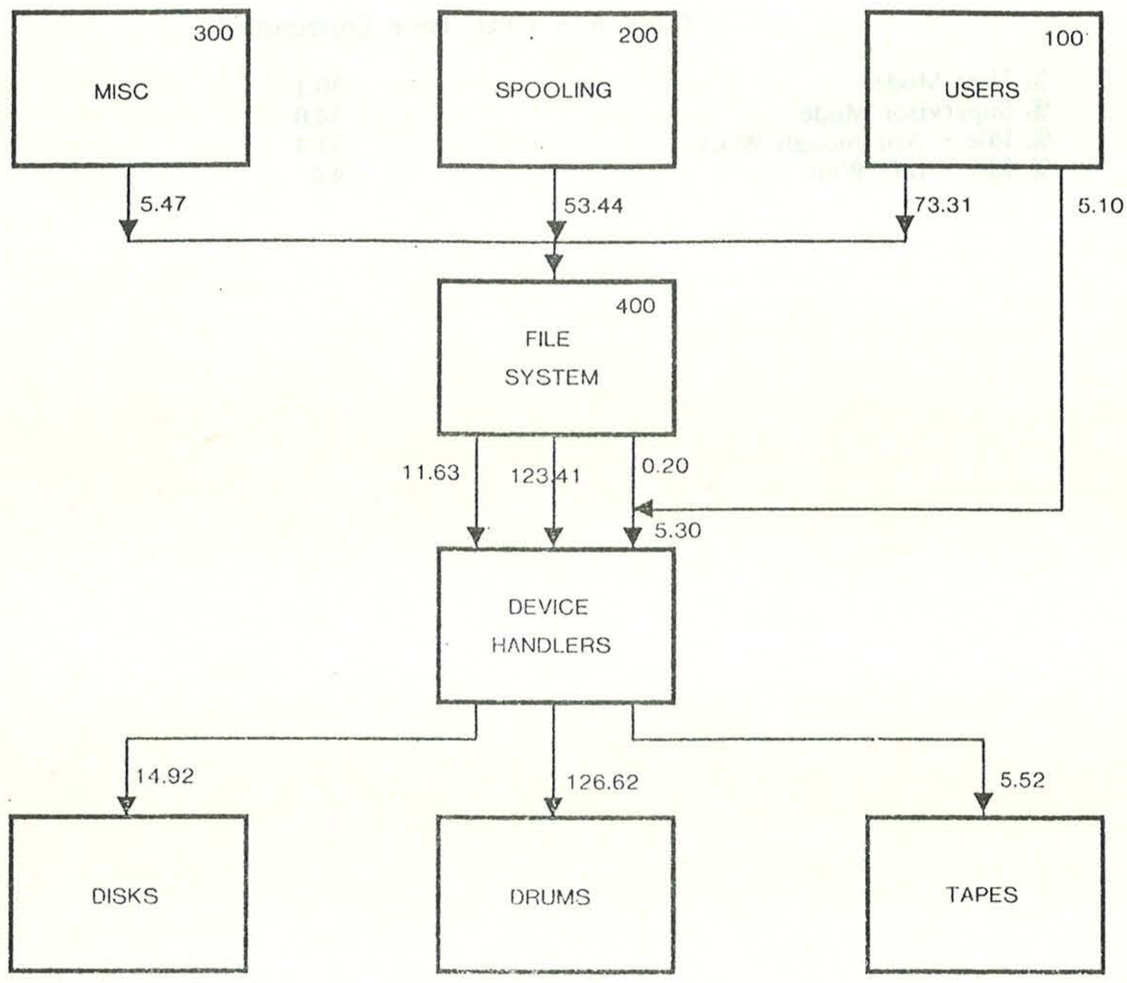
% User Mode	30.1
% Supervisor Mode	34.0
% Idle - Not enough Work	31.4
% Idle - I/O Wait	4.6

CHI/OS INPUT OUTPUT ACTIVITIES FLOW
10 AUG 76 @ 04:32:42 TO 10 AUG 76 @ 17:17:00

USER:CPU:REAL = 1.00:2.13:3.33

FIGURE 3

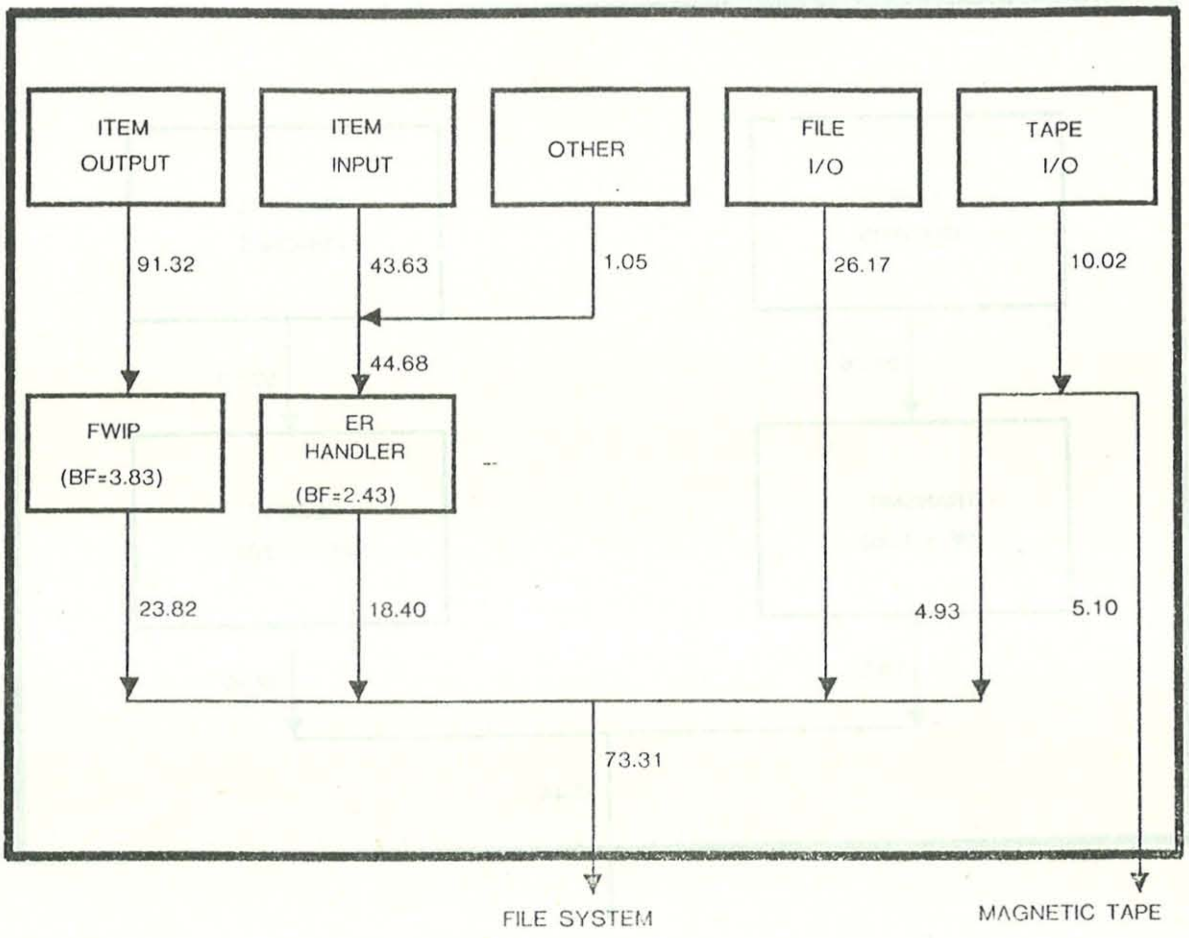
USER (100)



CHI/OS INPUT OUTPUT ACTIVITIES FLOW
10 AUG 76 @ 04:32:42 TO 10 AUG 76 @ 17:17:00
USER:CPU:REAL = 1.00:2.13:3.33

FIGURE 4

USER (100)



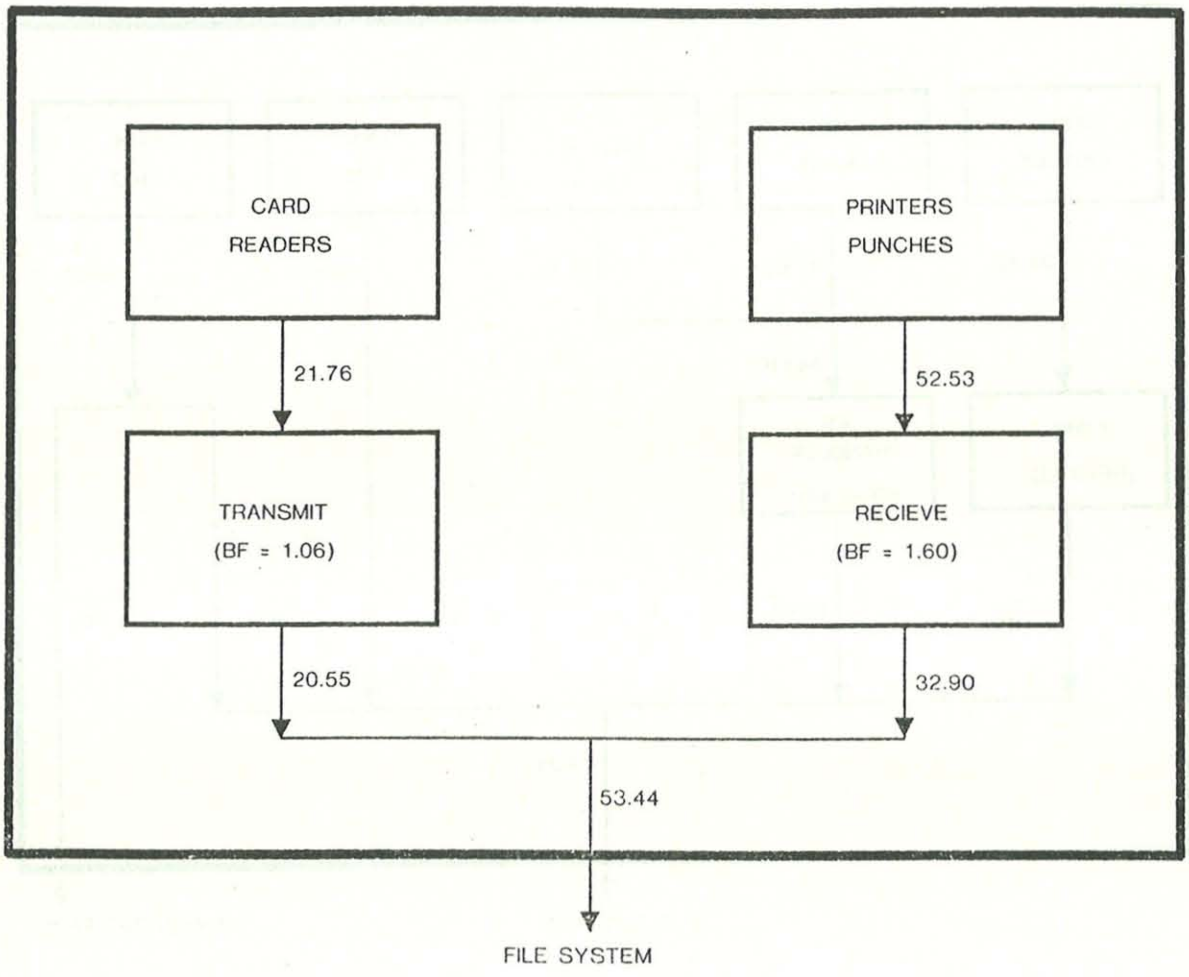
CHI/OS INPUT OUTPUT ACTIVITIES FLOW

10 AUG 76 @ 04:32:42 TO 10 AUG 76 @ 17:17:00

USER:CPU:REAL = 1.00:2.13:3.33

FIGURE 5

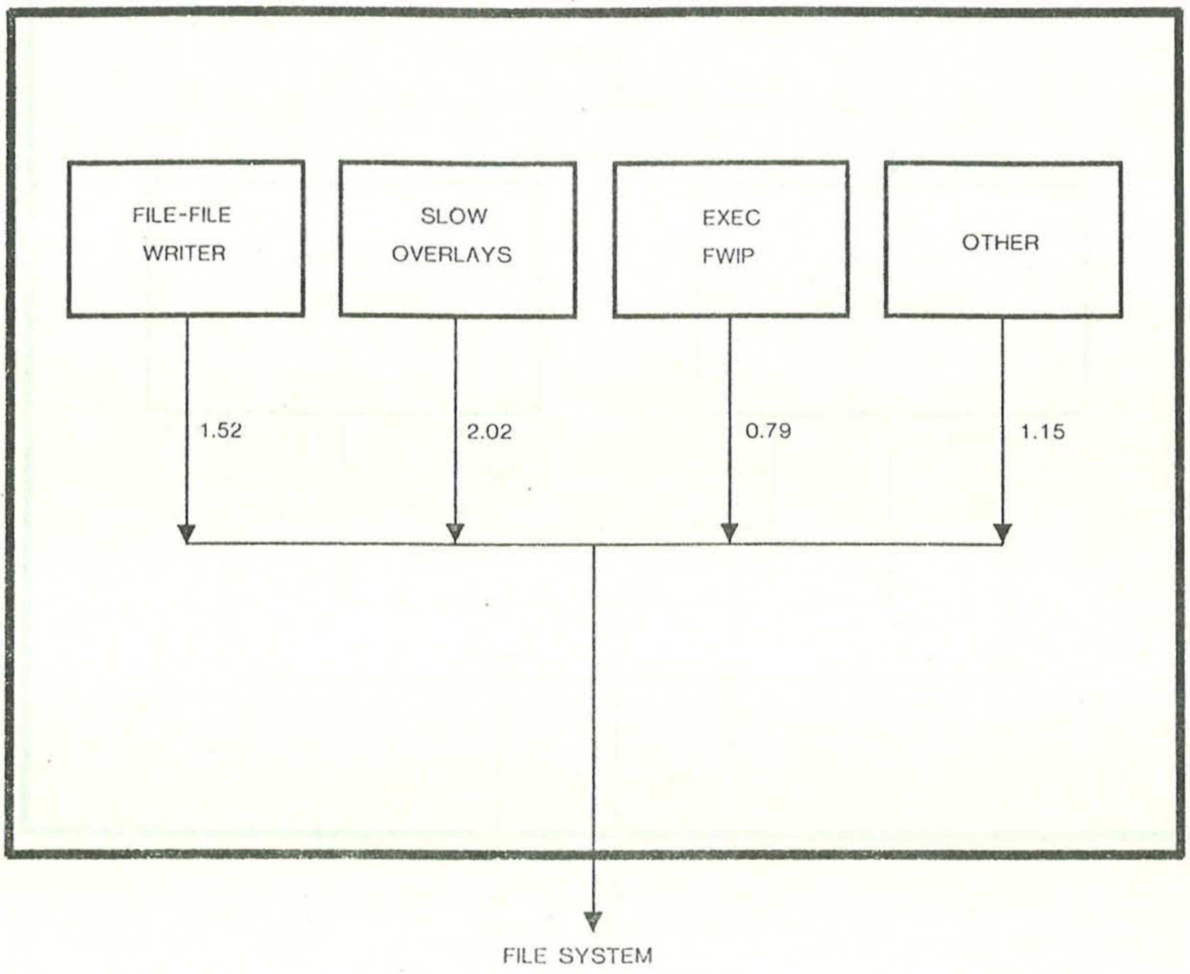
SPOOLING (200)



CHI/OS INPUT OUTPUT ACTIVITIES FLOW
10 AUG 76 @ 04:32:42 TO 10 AUG 76 @ 17:17:00
USER:CPU:REAL = 1.00:2.13:3.33

FIGURE 6

MISC PROCESSES (300)

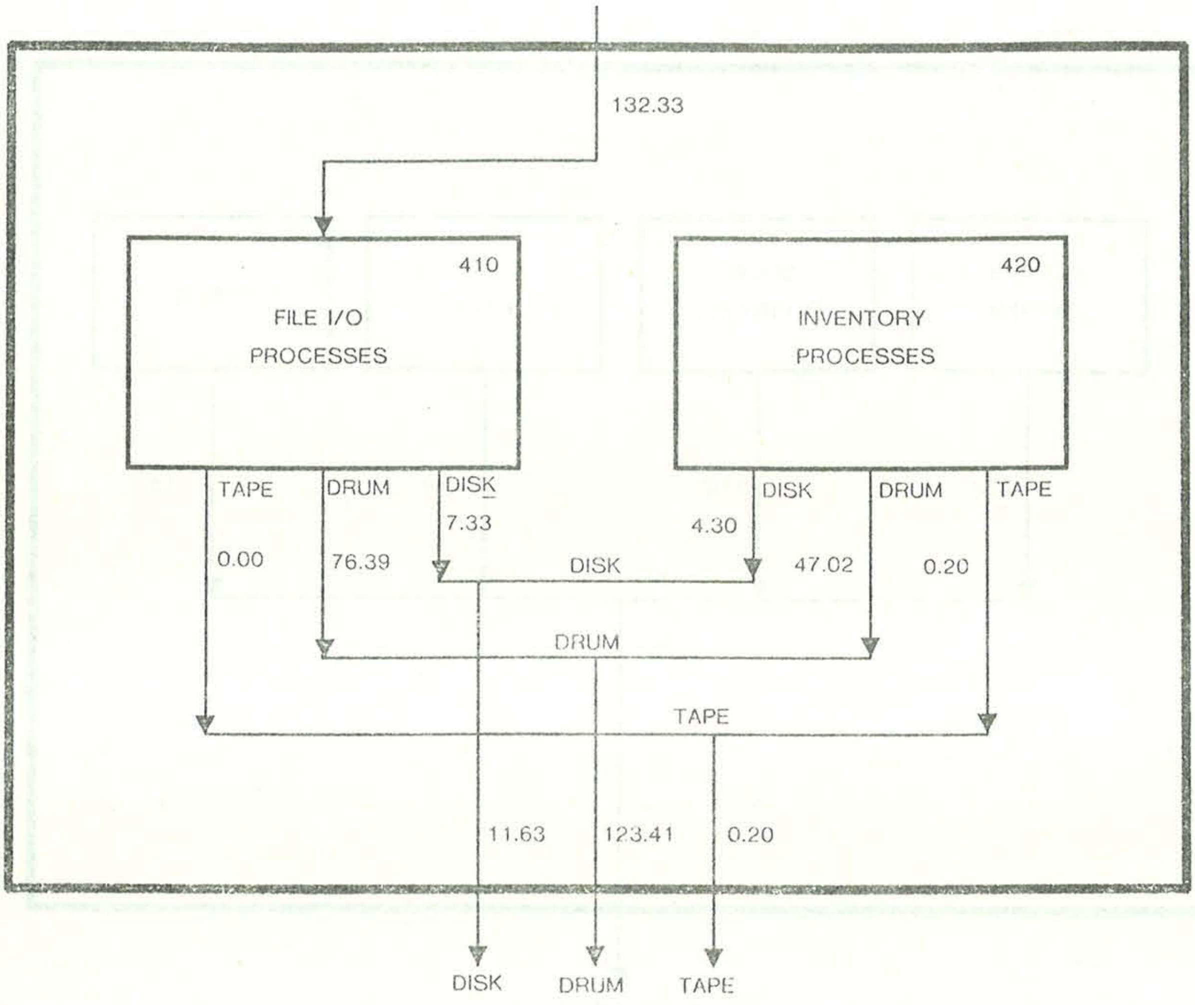


CHI/OS INPUT OUTPUT ACTIVITIES FLOW
10 AUG 76 @ 04:32:42 TO 10 AUG 76 @ 17:17:00
USER:CPU:REAL = 1.00:2.13:3.33

FIGURE 7

FILE SYSTEM

OVERVIEW (400)



CHI/OS INPUT OUTPUT ACTIVITIES FLOW

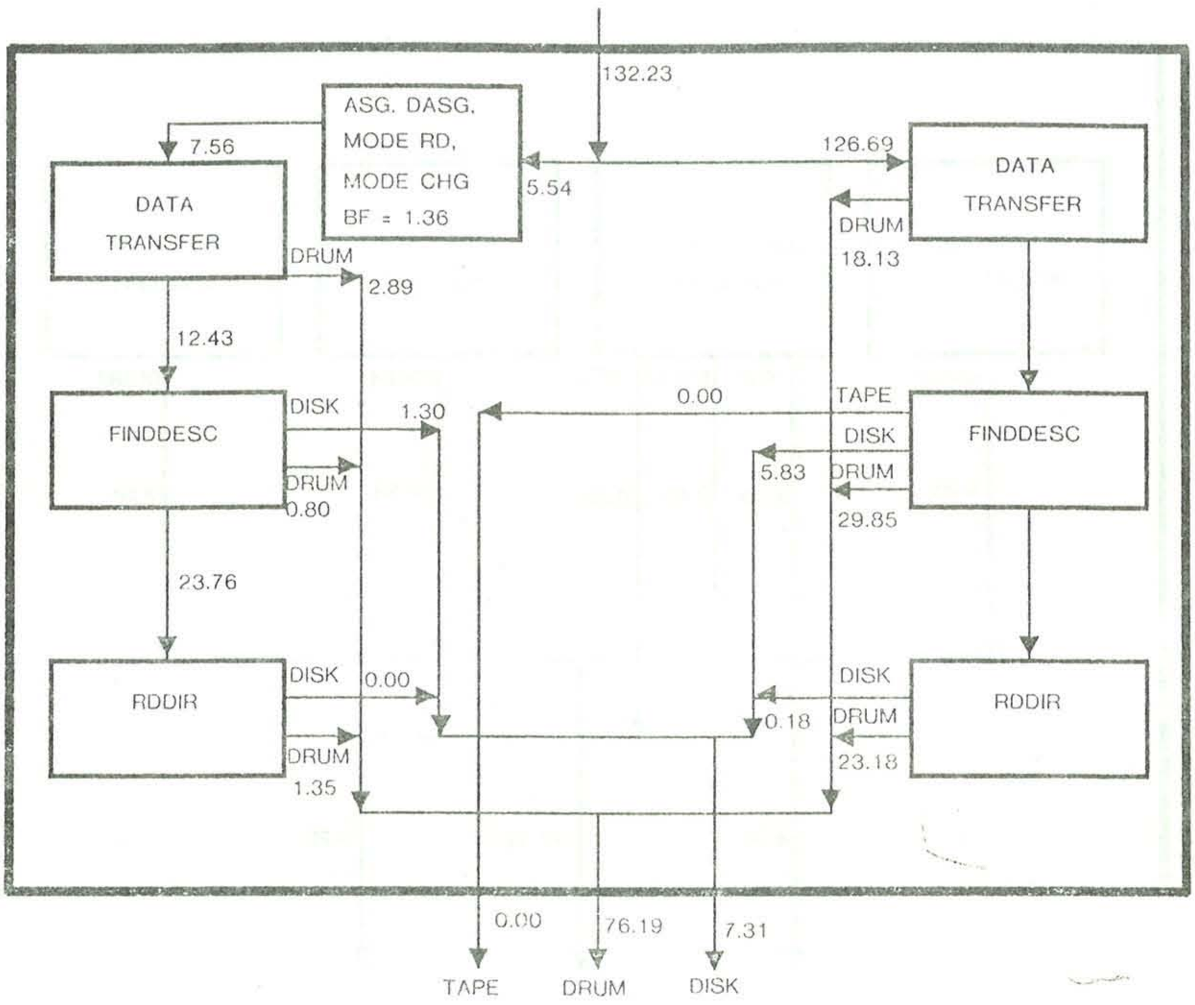
10 AUG 76 @ 04:32:42 TO 10 AUG 76 @ 17:17:00

USER:CPU:REAL = 1.00:2.13:3.33

FIGURE 8

FILE SYSTEM

FILE I/O PROCESSES (410)



CHI/OS INPUT OUTPUT ACTIVITIES FLOW

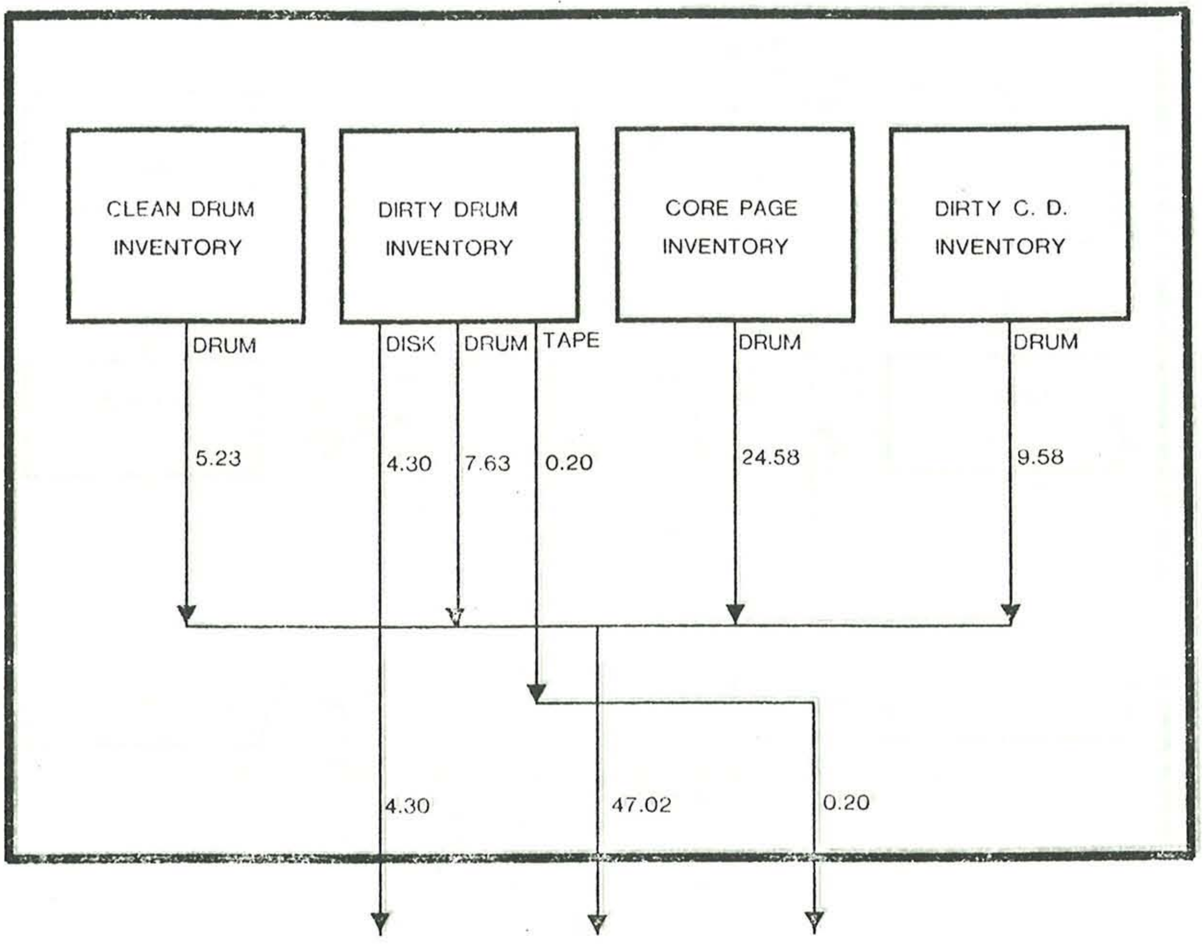
10 AUG 76 @ 04:32:42 TO 10 AUG 76 @ 17:17:00

USER:CPU:REAL = 1.00:2.13:3.33

FIGURE 9

FILE SYSTEM

INVENTORY PROCESSES (420)


CLEAN DRUM INVENTORY
DIRTY DRUM INVENTORY
CORE PAGE INVENTORY
DIRTY C. D. INVENTORY
DRUM
DISK
DRUM
TAPE
DRUM
DRUM
5.23
4.30
7.63
0.20
24.58
9.58
4.30
47.02
0.20