Explorer les codes et textes de MULTICS

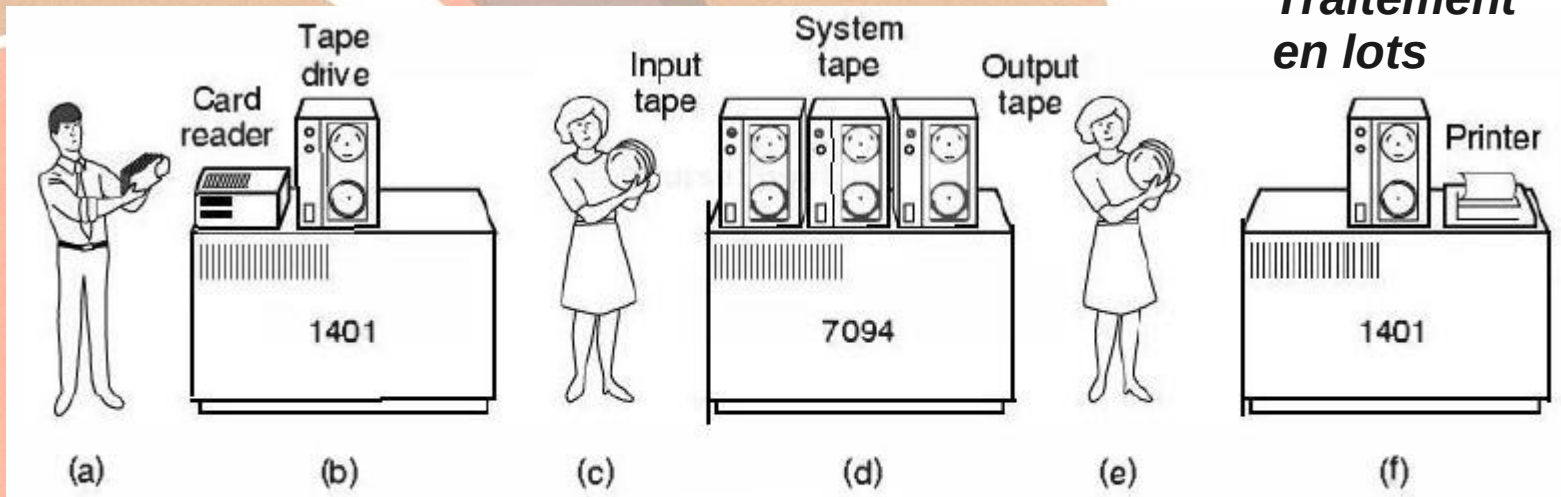
Maarten Bullynck (Paris 8 & UMR 7219 Sphère)

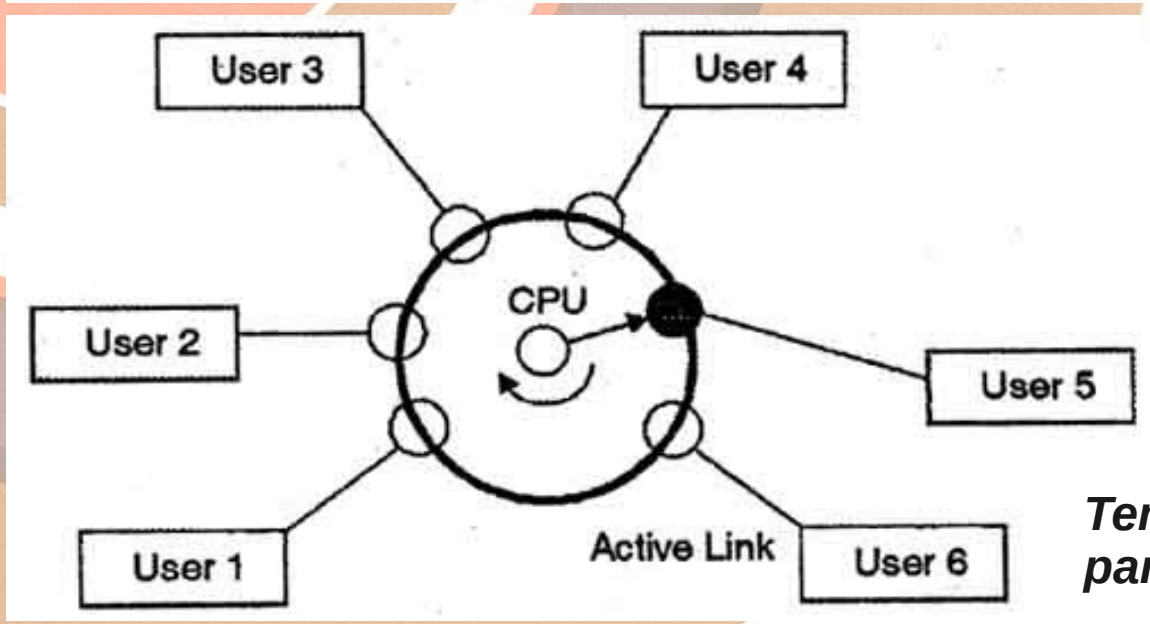
Systèmes d'exploitation: Quelques éléments d'histoire

- **La question: Qu'est-ce un système d'exploitation?**
- **Préhistoires**
  - *Preparatory routine* (1947, von Neumann & Goldstine, IAS machine)
  - *Utility programs, executive routines* (années 1950); *Comprehensive system of service routines* (Whirlwind, 1952)
- **Premiers systèmes d'exploitation** (*monitor systems, supervisor*) fin années 1950:
  - IBSYS sur IBM (1958)
  - Direct Input sur TX-0 (1958) ...
- **L'*Age classique* des systèmes d'exploitation**, années 1960
  - OS/360 pour l'IBM/360 (1964 ff)
  - Multics (MIT, GE, Bell) (1965 ff)
  - MCP, Burroughs (1966)
  - THE multiprogramming system, Dijkstra (1968)
  - Monitor, DEC (1968)
  - Unix (1969/71) ...

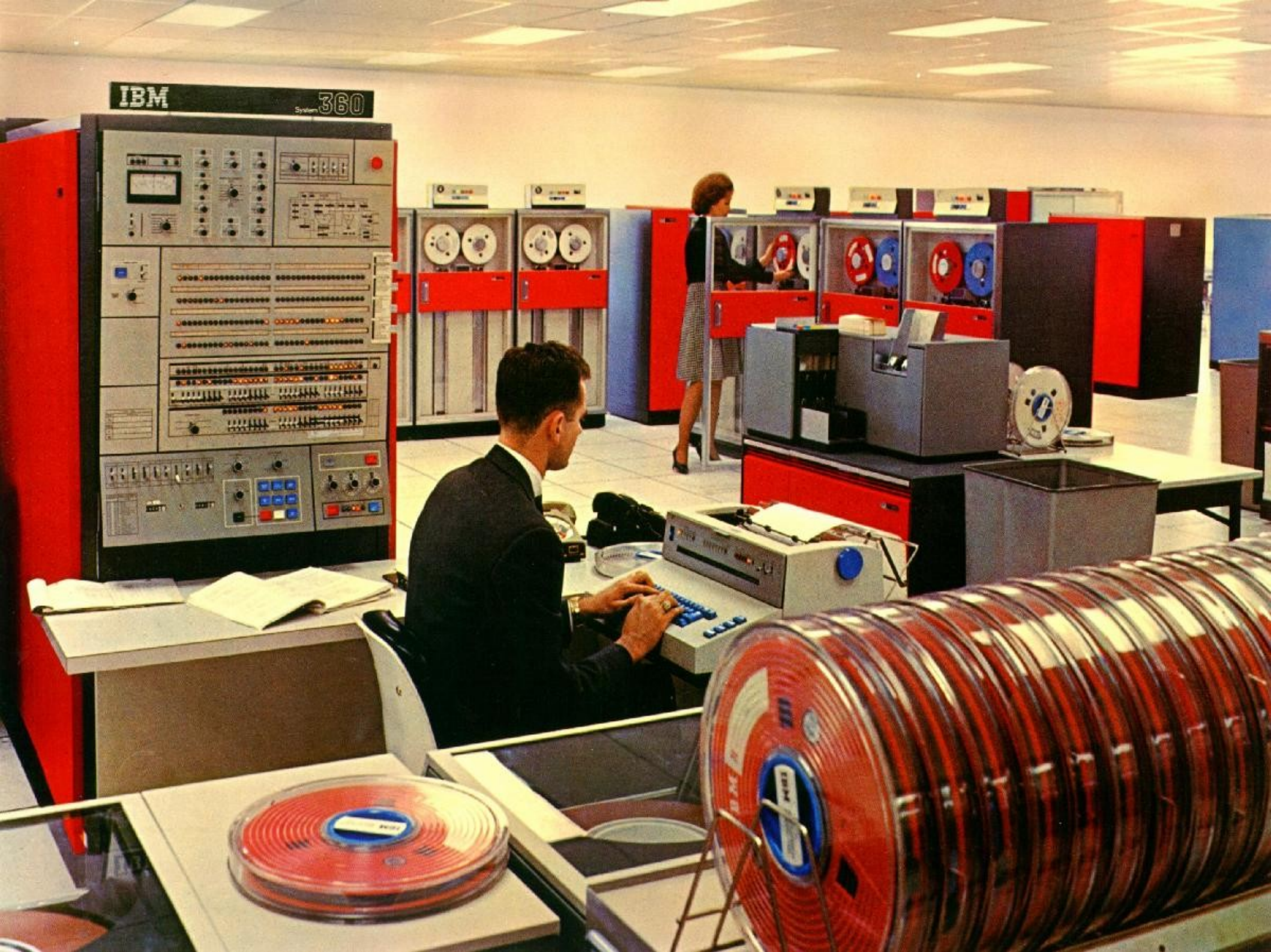
La discussion des années 1960: Batch processing vs. time sharing (traîtement en lots vs. temps partagé)

Traîtement en lots

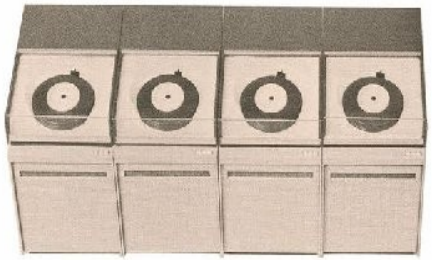




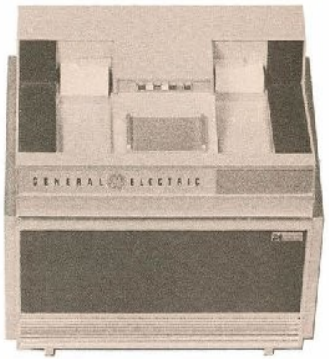
Temps partagé


IBM
System 360

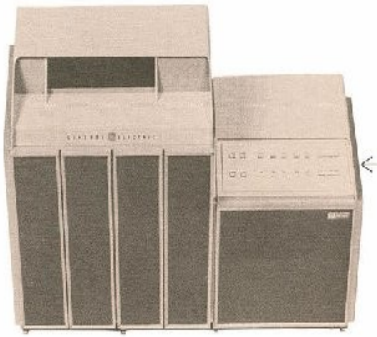
GE-400 Time-sharing System Configuration



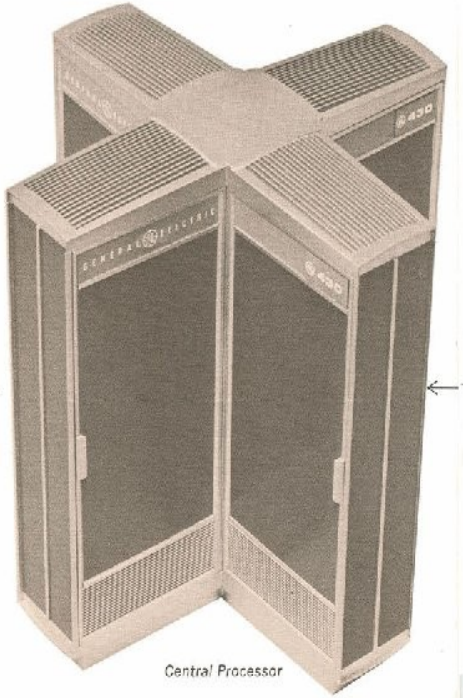
Removable Disc Storage Subsystem



Card Reader



Printer



Central Processor

Start With the GE-430 and Grow

The GE-430 is economical and powerful. It will support up to 30 simultaneous users. This means that approximately 120 terminals can be set up to logically serve 300-400 typical users. Add terminals as your demand increases. At the optimum time, you can switch to the GE-440.

The GE-440 can handle up to 50 users simultaneously. On a cost-per-channel basis, the GE-440 outperforms any other time-sharing system in its class.

As with all the GE-400's, growth is simple. All peripheral equipment is common; user programs are fully upward compatible. You save the cost of reprogramming.

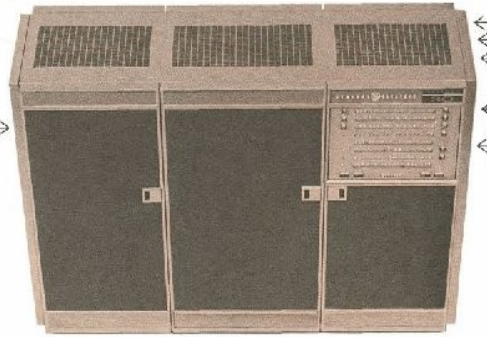
Here's what a typical GE-400 time-sharing system consists of:

Central processor
- Words of memory:
  32,000 for GE-430
  64,000 for GE-440
- Direct access package
- Time of day clock
- Floating point hardware
- Console

DATANET-30* communications processor
- 16,000 word memory
- Three bit buffer units (GE-430)
- Five bit buffer units (GE-440)
- Up to 10 channels per unit

Removable disc storage subsystem
- Four units and controller
- Stores over 30 million characters
- Transfers data at 208,000 characters per second
- Average random access time is 97.5 milliseconds

*DATANET, Reg. Trade-mark of General Electric Co.



DATANET-30 Communications Controller

Printer
- 600 lines-per-minute, or
- 1200 lines-per-minute

Card reader
- 600 cards-per-minute, or
- 900 cards-per-minute

Remote Terminals
- Uses models 33 and 35 Teletypewriter units
- Keyboard send/receive
- Automatic send/receive

Optional Equipment

The optional peripheral equipment below may be used when your system is not in the dedicated time-sharing mode of operation:
- Magnetic tapes — choose from over 12 subsystems depending on the speeds you need (tape units can be used in the time-sharing mode)
- Card punches — 100 and 300 cards-per-minute
- Fixed disc storage subsystem
- Removable cartridge mass storage subsystem
- Magnetic drum subsystem
- Perforated tape subsystem
- Reader/sorter subsystems — MICR reader/sorter and COC-5 document reader
- Multiple tape lister
- Peripheral switches

Model 33 or 35 Teletypewriter units

Quelques points de l'histoire de Multics

- L**'idée du temps partagé** (time-sharing, 1958-1960, Bemer, Strachey, McCarthy...), développer des modes interactifs d'utilisation et distribuer l'usage des ressources
- **CTSS** (*Compatible Time Sharing System*) sur un IBM 7094, développé par F. Corbató et collaborateurs à MIT, pour démontrer la faisibilité du partage de temps (1961-1963 ff)
- Project MAC (1963), financé par l'ARPA, promouvoir le **time-sharing**
- A partir de CTSS développer un système d'exploitation complet et complexe: **MULTICS** (*Multiplexed Information and Computing Service*) (1964-1969 ff)
  - Collaboration entre GE, MIT et Bell Labs, sur un ordinateur GE-645
  - 1965 Fall Joint Computer Conference, présentation de 6 papiers
  - PL/1 comme langage de programmation
  - 1967, anneaux de protection
  - 1969 première version sort
  - 1969, Bell Labs sort du projet; 1970 GE vend ses activités à Honeywell/Bul
  - 1974 Access Isolation Mechanism (AIM) ajouté et développement d'un noyau de sécurité
  - 1985, Multics recoit B2-certificat de sécurité
  - Une trentaine d'installations en France dans les années 1980 via Bull
  - Dernières installations (surtout militaires) fermées autour de 1995

Les langages de Multics

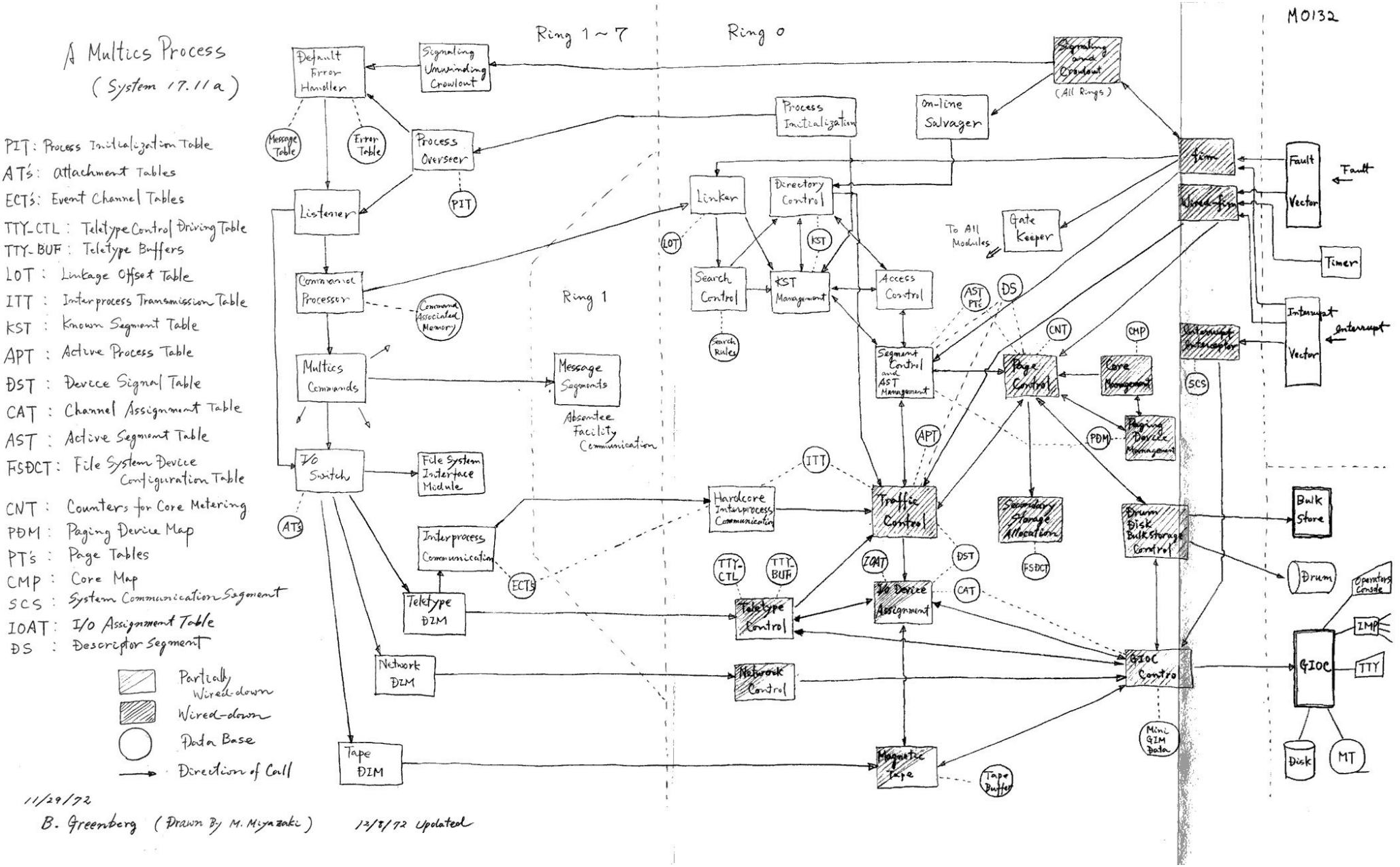
- **PL/1**, langage impératif de haut niveau, était choisi comme langage d'implémentation
  - PL/1 était développé par IBM depuis 1964 comme *general-purpose language* pour remplacer FORTRAN sur les machines IBM/360 (inspiration de ALGOL)
- Pour certaines parties du noyau, l'assembleur **ALM** (*Assembly Language for Multics*) était utilisé
  - Sans macros jusqu'en 1977
  - Surtout pour *page control, traffic control* et *boot,* pour déclarer les “base de données” des programmes
  - Assembleur du GE 645 avec microprogrammation
- Certains programmes étaient en BCPL (Basic Combined Programming Language), hérité de CTSS et développé par Bell Labs
- En général (Huber 1976):
  - longueur[programme en ALM] = 2 x longueur[programme en PL/1]
  - Après compilation en code machine:

    2 x longueur[programme en ALM] = longueur[programme en PL/1]

La documentation de Multics

- Articles parus dans des revues
- ***Multics programmer manual*** (MPM)
  - *Reference Guide* (214pp)
  - *Commands and Active Functions* (891pp)
  - *Subroutines* (1566pp)
  - *Subsystem Writers' Guide* (537pp)
  - *Peripheral Input/Output* (188pp)
  - *Communications Input/Output* (178pp)
- Manuels sur les langages PL/1, APL,ALM etc.
- Les rapports documentation le développement de Multics, *Multics Technical Bulletin*
- Aujourd'hui, beaucoup en ligne sur multicians.org et bitsavers.org

Structure générale de Multics (1974)



Code dans le noyau de Multics (1974)

CATEGORY	MODULES	SIZE	NON-PL/I	TEXT-LEN
Initialization	45	4636	37%	23708
Reconfiguration	13	1126	2%	7714
Fault Handling	13	1326	90%	2158
I/O Control	38	3526	28%	17598
Printer	6	958	73%	2532
Tape Control	21	2662	2%	10449
TTY Control	19	4266	5%	20477
ARPANET	55	7493	1%	40338
Error Handling	25	2016	9%	9312
Process Control	28	1296	6%	8773
Traffic Control	3	2296	92%	2710
IPC	25	3061	2%	16160
Process Signals	5	390	17%	1450
Resource Control	32	2343	---	11342
Storage System	38	5366	1%	35864
Directory Control	51	6609	< 1%	34434
Segment Control	32	1973	5%	11460
Page Control	26	5870	69%	13704
Salvager	18	2897	1%	20747
Dynamic Linker	14	1793	11%	7234
File System	5	1161	2%	6631
AIM	7	924	6%	6223
Error Interpretation	12	856	1%	7228
Kernel Utility	5	470	93%	663
Shared Utility	16	2800	81%	5069
Backup	44	7827	---	57002
Answering Service	73	12987	2%	94609
PL/I Support	39	12504	94%	18641
Miscellaneous	4	191	22%	918
Totals	712	101623	26%	495148
(Ring 0 Only)	432	61848	41%	232824

Le développement de l'ordonnanceur de Multics (scheduler)

- Développement théorique
  - J. Salzer *Traffic Control in a Multiplexed Computing System* (1966, Thèse)
  - R. Rappaport, *Implementing Multi-Process Primitives in a Multiplexed Computer System* (1968, Thèse)
  - R. Mullen, *Priority Scheduler*, MTB-193 (1975)
- Développement pratique
  - CTSS, "Greenberger-Corbató exponential scheduler", 1965
  - Multics: **pxss.alm**, Process Exchange Switch Stack
    - Scheduler, 1e version, 1967
    - Workclass Scheduler, 1975
    - Deadline Scheduler, 1976

```
" " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " "
"
"   pxss -- The Multics Traffic Controller (Scheduler)
"
"   Last Modified: (Date and Reason)
"
"   April 1983 by E. N. Kittlitz to DRL instead of 0,ic looping.
"   [...]
"   Winter 1977 RE Mullen for lockless (lockfull?) scheduler:
"       concurrent read_lock, ptlocking state, apte.lock,
"       unique_wakeup entry, tcpu_scheduling
"   Spring 1976 by RE Mullen for deadline scheduler
"   02/17/76 by S. Webber for new reconfiguration
"   3/10/76 by B Greenberg for page table locking event
"   Spring 1975 RE Mullen to implement priority scheduler and
"       delete loop_wait code.  Also fixed plm/lost_notify bug.
"   Last modified on 02/11/75 at 19:49:10 by R F Mabee.  Fixed arg-copying &
other bugs.
"   12/10/74 by RE Mullen to add tforce, ocore, steh, tfmax & atws
"        disciplines to insure response in spite of long quanta, and
"        fix bugs in get_processor, set_newt, and loop_wait unthreading.
"   12/6/74 by D. H. Hunt to add access isolation mechanism checks
"   4/8/74 by S.H.Webber to merge privileged and unprivileged code.
"       and to add quit priority and fix lost notify bug
"   5/1/74 by B. Greenberg to add cache code
"   8/8/72 by R.B.Snyder for follow-on
"   2/2/72 by R. J. Feiertag to a simulated alarm clock
"   9/16/71 by Richard H. Gumpertz to add entry rws_notify
"   7/**/69 by Steve H. Webber
"
" " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " "
```

Pxss.alm preamble

```
"SCDA8C   BCD 07/23/70 1553.6   19706         00000
1       SCDA   ************ TIME SHARING SCHEDULING ALGORITHM ************    07/22/70 1321.5
PAGE 1

                          *                          T. HASTINGS AND R. DALEY
  SCDA0002
                          *
  SCDA0003
                          *              MINOR MODIFICATIONS BY G. SCHROEDER WHEN NEW
  SCDA0004
                          *               I/O PACKAGE INSTALLED....SUMMER,1965.
  SCDA0005
                          *              MINOR CHANGES FOR NEW COMMAND PROCESSOR
  SCDA0006
                          *               SPRING 1969 ...... P.R. BOS
  SCDA0007
                          *
  SCDA0008
                          *          THE SCHEDULING ALGORITHM PERFORMS THE FOLLOWING FUNCTIONS
  SCDA0009
                          *
  SCDA0010
                          *           1. DETERMINES WHICH USER IS TO RUN NEXT
  SCDA0011
                          *           2. DETERMINES WHEN NEXT USER IS TO RUN
  SCDA0012
                          *           3. DETERMINES HOW LONG NEXT USER IS TO RUN
  SCDA0013
                          *           4. CHARGES USERS FOR SWAPPING AND RUNNING TIME
  SCDA0014
                          *           5. KEEPS TRACK OF THE STATUS OF EACH USER
  SCDA0015
" " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " "
```

CTSS Time sharing scheduling algorithm, preamble

In order to optimize the response time to a user's command or program, the supervisor uses a multi-level scheduling algorithm. The basis of the algorithm is the assignment of each program as it enters working or waiting command status to an nth level priority queue. Programs are initially entered at a level which is a function of the program size (i.e. at present, programs of less than 4k words enter at level 2 and longer ones enter at level 3). There are currently 9 levels (0-8). The process starts with the supervisor operating the program which is first in the queue at the lowest occupied level, L. the program executes for a time limit = P.L quanta; a quantum of time is one half second. If the program has not finished (left working status) by the end of the time limit, it is placed at the end of the next higher level queue. The program at the head of the lowest occupied level is then brought in. If program P enters the system at a lower level than the program currently running, and if the current program P1 has run at least as long as P is allotted, then P1 will be returned to the head of its queue and P will be run.

CTSS Time sharing scheduling algorithm, 1969

The seven calls can be classified into three groups as follows:

1. Process Wait and Notify (PWN) calls: wait, notify.
2. Interprocess Communication (IPC) calls: block, wakeup.
3. Process interrupt calls: re-schedule, pre-empt, stop.

The Process Wait and Notify Calls

Every process reaches a point in its execution where it has to have information from some other process; if the information is unavailable, it abandons the processor on which it currently executes until such time as the information will become available, or until that event happens. We name "event" anything that is observed by some other process and which is of interest to our process. We distinguish between two classes of events, system-events and user-events. This distinction is made to take efficient advantage of known characteristics of system events. (In principle, events are all of the same nature and can be handled uniformly.)

Les procès définis par Salzer dans sa thèse, 1966

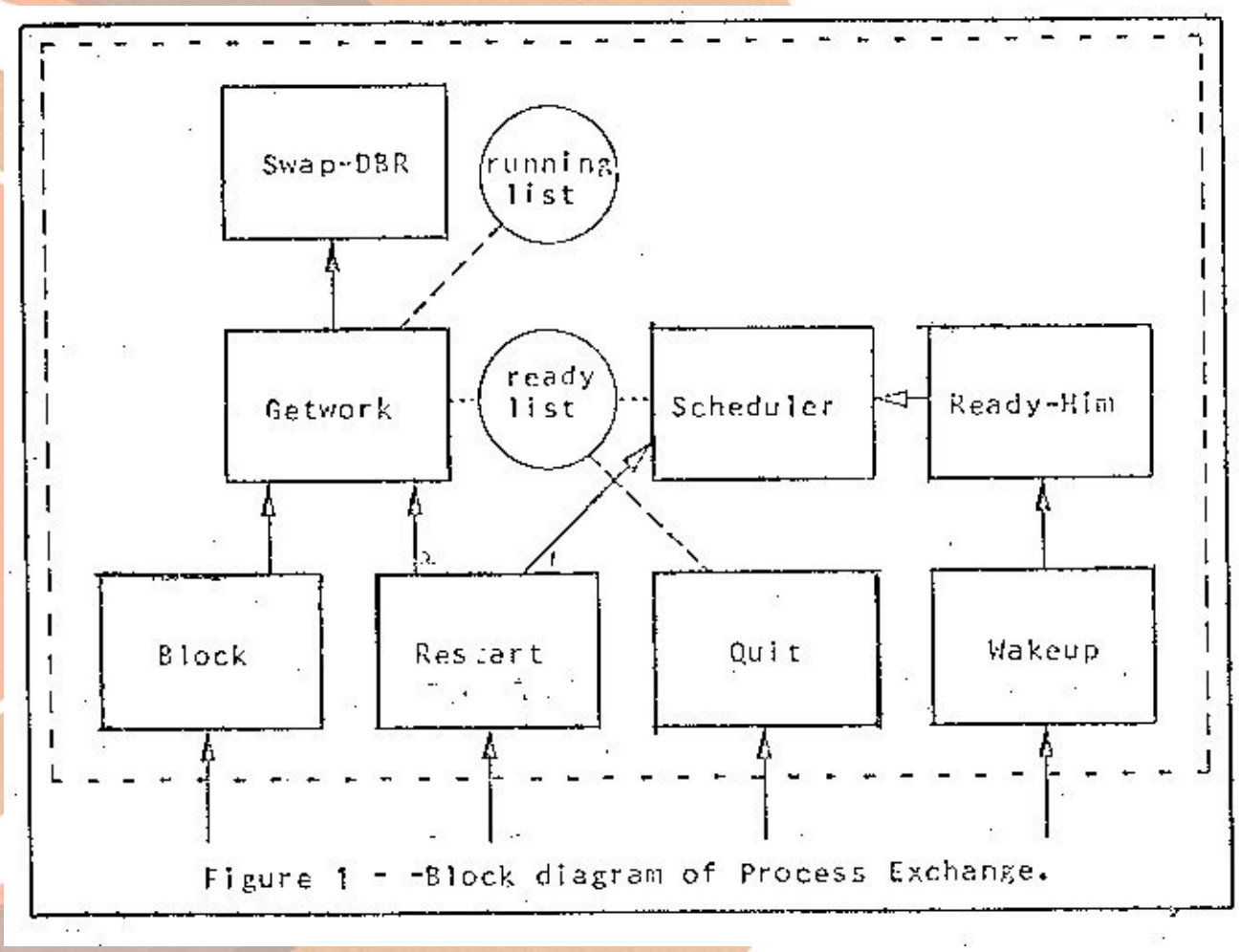


Figure 1 - -Block diagram of Process Exchange.

Structure générale par Salzer dans sa thèse, 1966

A computer system is a vehicle in which various tasks or processes are executed. In all computer systems, at least two primitive process control functions exist. These are:
1. The ability to create or introduce new processes. We will call this the process creation primitive.
2. The ability to forceably halt the execution of a process. This ability rests in some force or power outside the process (possibly in another process. We will call this the process destruction primitive.
3. The ability for a process to declare that it has finished and ought to be terminated. We will call this the suicide primitive.

In his PhD, Salzer proposed to add 4 primitives:
1. The block primitive which includes suicide.
2. The wake up primitive.
3. The reschedule primitive (originally named restart by Saltzer) .
4. The stop primitive (originally named quit by Saltzer) .
These four primitives make up what Saltzer calls the Process Exchange.

The Process Wait and Notify (PWN) facility offered its users four entry points: addevent , delevent , wait , and notify . Addevent allowed a process to allocate an entry and to thread the entry onto a particular list. Delevent allowed a process to unthread itself from a list and deallocate its entry. Wait allowed a process to check that it was still on a given list. If the process was still on the list wait called block. If not, wait returned. Notify allowed a process to pick up an entire list, call wakeup for each process on the list and unthread the entries from the list.

Procès dans la thèse de Rappaport, 1968

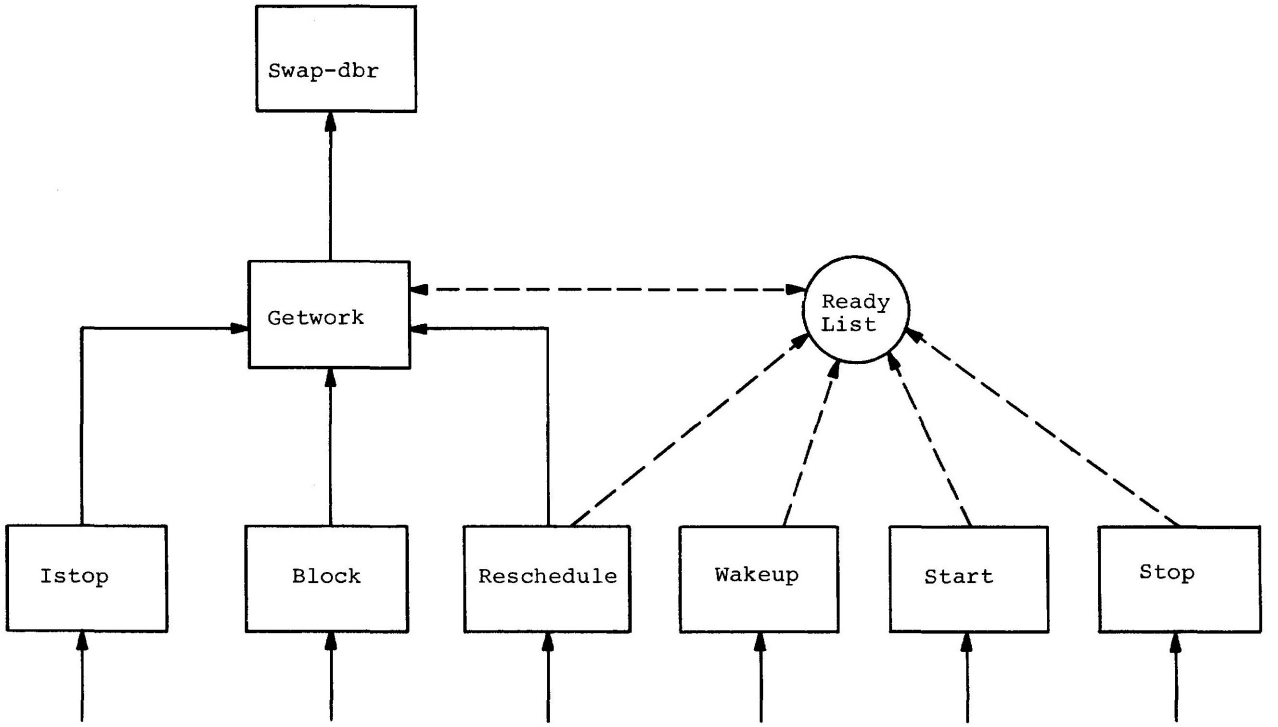


Figure 4. Block Diagram of Simple Traffic Controller. Solid Lines Indicate Closed Subroutine Calls Whereas Dashed Lines Indicate Data Flow.

Structure générale par Rappaport dans sa thèse, 1968

This MTB proposes that the scheduler allow the grouping of processes into work classes and provide each work class with a guaranteed percentage of available cpu time. Conceptually each work class will be assigned a virtual processor […]
The actual algorithm used to enforce the proper sharing of the cpu resource will be as follows: Imagine the existence of a system virtual clock which increments as virtual time is used by non-idle processes. Imagine also that each work class has a store of credits (in units of microseconds) which is continually growing at a rate proportional to the speed of the virtual clock multiplied by the fraction of cpu resources which the work class is to receive. Suppose further that the store of credits for the work class is decremented as members actually consume virtual cpu time. Clearly it is undesirable to allow credits to build up indefinitely for a work class with no processes ready, so a maximum value is set on the number of credits which can be accumulated. In addition the value is restricted from ever becoming negative. The algorithm for chosing the next work class from which to choose a process to which to award eligibility may then be as simple as choosing that work class which has accumulated the maximum number of credits.

Multics Workclass Scheduler, 1975

In 1975-1976, Bob Mullen added features to the scheduler to provide more efficient support for daemons driving physical devices such as printers. allow precise tuning of workloads for competitive benchmarks. The version of the scheduler is called the "Deadline Scheduler."
The deadline scheduler used the workclass structures to implement a wholly different scheduler in which neither the FB-n nor the percentage parameters were used. Considering that most people do not understand the FB-n algorithm, the top level view was that the scheduler could be operated in either "percent" mode or "deadline" mode.
Using some workclasses with virtual deadlines along with a few with realtime deadlines was a convenient way to tune benchmarks with response time requirements which varied for different user scripts.

Multics deadline scheduler, 1976

The traffic control subsystem is implemented almost entirely by one large ALM program named pxss. Various entries into pxss are used by other ring-0 procedures to change the state of the current process or to send a wakeup signal to some other process. Most of pxss runs in a wired environment with interrupts masked, in order to protect critical data bases (most notably the wired segment tc_data).

Without a theory of computing systems to fall back on,designing of such complex systems becomes an art, rather than a science, in which it is impossible to prove the degree to which working solutions to problems are in any sense optimum solutions. In much the same way as authors write books, large computer systems go through several drafts before they begin to take shape. In the absence of a theory one can only cope with the complexity of the situation by proceeding in an orderly fashion to first produce an initial working model of the desired system. This part of the work represents the major effort of the design and implementation project. Once having arrived at this benchmark, many of the problems may then be seen in a clearer light and revisions to the working model are implemented much more quickly than were the original modules. As to the development of a theory, one gets the impression that it will be a long time in coming.

Conclusion de la thèse de Rappaport, 1968