Published: 05/25/67

Identification

Ring register simulation module
R. C. Daley, D. M. Ritchie

Purpose

Currently the ring mechanism for protection of the supervisor (see BD.9) is implemented by using a different descriptor segment for each ring. At some future date the 645 may include a ring register and additional hardware to interpret an access bracket field within each SDW of a single descriptor segment. This section describes certain procedures which aid in fostering the illusion that the ring hardware is on the present 645, and which thus minimize the software changes necessary when a ring register is actually available.

Introduction

On an as yet hypothetical 645 with a ring register, the contents of this register are stored with the SCU data obtained at the time of a fault or interrupt, and also by means of a "store ring register" instruction. Conversely, an RCU instruction given at the conclusion of the treatment of a fault or interrupt loads the ring register with a possibly new ring number, as does a "load ring register" instruction.

Thus if a ring register is to be simulated the fault and interrupt interceptors must be able to determine what ring the process was operating in at the time of the fault or interrupt so as to be able to simulate the ring number portion of the SCU data; conversely, when the return to the faulting or interrupted procedure takes place, the interceptors must be able to call for a change of rings depending on the simulated ring register stored with the SCU data.

The routine described below provide for these needs.

Procedures

The procedures ring$store and ring$load described below are callable only in master mode from the fault and interrupt interceptors. They are EPLBSA coded master mode procedures.

The procedure setup_ring is an EPL procedure called only by the ring-crossing fault handler and by the process load module routine loadproc2.

1. ring$load

The statement

```
call ring$load (ringno, error);
```

executes the LDBR instruction that causes the process to switch to ring number ringno. It thus simulates a "load ring register" instruction. Then the SDW for the processor stack belonging to the processor being used is copied from the old descriptor segment to the new descriptor segment.

The descriptor segment for the desired ring must have its first page in wired-down core. The second argument error is set to non-zero if this is not the case. If the change was successful, error is set to zero.

2. ring$store

The statement

```
call ring$store (ringno);
```

places in ringno the number of the current ring. It thus simulates a "store ring register" instruction.

3. setup_ring

The statement

```
call setup_ring (ringno);
```

causes a descriptor segment for ring number ringno to be constructed and its first page placed in wired-down core.

In any loaded process, the hardcore ring descriptor segment always has its first page wired down. In addition, there is at most one other wired-down descriptor segment.

Therefore, if setup_ring is called with a ring number whose descriptor segment is not wired down, and if there is another non-hardcore descriptor segment whose first page is wired down, setup_ring will release the latter segment from its wired down state before wiring down the first page of the new descriptor segment.

This strategy means that control may pass from an outer ring to the hardcore ring and back without the need for creating any new descriptor segments, and thus without calling setup_ring. Only when the hardcore supervisor desires to switch control to a ring other than the one from which it was entered is the use of set_up ring necessary; in fact for a loaded process, the latter situation occurs only during the treatment of a ring-crossing fault. The remaining occasion when setup_ring is used occurs during the loading of a process (see BG.3.03).

Implementation Notes

Information regarding the status of rings is kept in the Process Data Segment (see BJ.1.03). The following variables are involved.

1. hardcore_ring and hardcore_dbr

   These are the number of and the DBR value for the hardcore ring and its descriptor segment. hardcore_dbr is changed only by the Process Load Module routine loadproc (BG.3.03).

2. cur_ring

   This variable always contains the ring in which the process is currently operating. It is modified only by load$ring.

3. wired_ring and wired_dbr

   If the first page of a descriptor segment for any non-hardcore ring is wired down, wired_ring contains the number of this ring and wired_dbr contains the corresponding DBR value. If wired_ring equals hardcore_ring, then the descriptor segment for the hardcore ring alone is wired down, and wired_dbr is meaningless. Wired_ring and wired_dbr are changed only by setup_ring.

Thus, if setup_ring is called with argument ringno ≠ hardcore ring, then if ringno equals wired ring, do nothing. But if ringno does not equal wired ring, create a descriptor segment for ring ringno. Then if wired ring is not equal to hardcore ring, unwire wired ring. In any case set wired ring equal to ringno.