DIRECTORY
ControlDefs: FROM "controldefs";
Mopcodes: FROM "mopcodes";

ProcessDefs: DEFINITIONS
BEGIN
SOC: PRIVATE POINTER TO CARDINAL = LOOPHOLE[20B];
CurrentPSB: PRIVATE POINTER TO ProcessHandle = LOOPHOLE[21B];
ReadyList: PRIVATE QueueHandle = LOOPHOLE[22B];
CurrentState: PRIVATE POINTER TO ControlDefs.SVPointer = LOOPHOLE[23B];

DIW: POINTER TO WORD = LOOPHOLE[421B];
WakeupsWaiting: PRIVATE POINTER TO WORD = LOOPHOLE[452B];
ActiveWord: PRIVATE POINTER TO WORD = LOOPHOLE[453B];

InterruptLevel: TYPE = [0..15];
ParityLevel: InterruptLevel * 0;
SwatLevel: InterruptLevel * 3;
TimeoutLevel: InterruptLevel * 4;
UnusableLevel: InterruptLevel * 15;
ConditionVector: TYPE = ARRAY InterruptLevel OF POINTER TO CONDITION;
CV: POINTER TO ConditionVector = LOOPHOLE[40B];

MonitorLock: TYPE = MACHINE DEPENDENT RECORD
  lock: {locked, unlocked},
  -- priority: Priority,
  queue: PackedQueue;

MonitorHandle: TYPE = POINTER TO MonitorLock;
LockedEmpty: MonitorLock = [locked, Empty];
UnlockedEmpty: MonitorLock = [unlocked, Empty];

-- NOTE: Both fields of a MonitorLock are packed into the same word, with
-- the lock in the high-order bit and "locked" represented by zero, so
-- that a MonitorHandle to a locked MonitorLock can be loopholed into a
-- QueueHandle.

Condition: TYPE = MACHINE DEPENDENT RECORD
  wakeupWaiting: {no, yes},
  queue: PackedQueue,
  timeout: Ticks;

ConditionHandle: PRIVATE TYPE = POINTER TO Condition;

-- NOTE: The first two fields of a Condition are packed into the same word,
-- with wakeupWaiting in the high-order bit and "no" represented by zero,
-- so that a ConditionHandle to a Condition without a waiting wakeup can
-- be loopholed into a QueueHandle.

Fork: PROCEDURE [UNSPECIFIED] RETURNS [ProcessHandle];
Join: PROCEDURE [ProcessHandle] RETURNS [ControlDefs.FrameHandle];
Detach: PROCEDURE [UNSPECIFIED];
ValidateProcess: PROCEDURE [ProcessHandle] RETURNS [ControlDefs.FrameHandle];
GetPriority: PROCEDURE RETURNS [Priority];
SetPriority: PROCEDURE [Priority];
SetTimeout: PROCEDURE [condition: POINTER TO CONDITION, ticks: CARDINAL];
DisableTimeout: PROCEDURE [POINTER TO CONDITION];
Abort: PROCEDURE [UNSPECIFIED];
EnableScheduling, DisableScheduling, Yield: PROCEDURE;

InitializeMonitor: PROCEDURE [monitor: POINTER TO MONITORLOCK];
InitializeCondition: PROCEDURE [condition: POINTER TO CONDITION, ticks: CARDINAL];

TooManyProcesses: ERROR;
Aborted, TimedOut: SIGNAL;

ProcessHandle: PRIVATE TYPE = POINTER TO PSB;

PSB: TYPE = PRIVATE MACHINE DEPENDENT RECORD
  link: ProcessHandle,
cleanup: ProcessHandle,
timeout: Ticks,
enterFailed: BOOLEAN,
detached: BOOLEAN,
fill: [0..37B],
state: {frameReady, frameTaken, dead, alive},
timeoutAllowed, abortPending, timeoutPending, waitingOnCV: BOOLEAN,
priority: Priority,
frame: ControlDefs.FrameHandle];

Priority: TYPE = [0..7];
DefaultPriority: Priority = 1;

TimerGrain: CARDINAL = 50; -- 50 milliseconds/tick
Ticks: TYPE = CARDINAL;
DefaultTimeout: Ticks = 100;
MsecToTicks: PROCEDURE [CARDINAL] RETURNS [Ticks];
TicksToMsec: PROCEDURE [Ticks] RETURNS [CARDINAL];

Clean: PRIVATE ProcessHandle = LOOPHOLE[0];

NullQueueHandle: PRIVATE QueueHandle = LOOPHOLE[0];
QueueHandle: PRIVATE TYPE = POINTER TO Queue;
Queue: PRIVATE TYPE = ProcessHandle;
PackedQueue: PRIVATE TYPE = POINTER [0..77777B] TO PSB;
Empty: PRIVATE PackedQueue = FIRST[PackedQueue];

Enter: PROCEDURE [POINTER TO MONITORLOCK] RETURNS [success: BOOLEAN] •
MACHINE CODE BEGIN Mopcodes.zME END;
Exit: PROCEDURE [POINTER TO MONITORLOCK] =
MACHINE CODE BEGIN Mopcodes.zMXD END;
Wait: PROCEDURE [POINTER TO MONITORLOCK, POINTER TO CONDITION, CARDINAL] =
MACHINE CODE BEGIN Mopcodes.zMXW END;
ReEnter: PROCEDURE [POINTER TO MONITORLOCK, POINTER TO CONDITION] RETURNS [success: BOOLEAN] = MACHINE CODE BEGIN Mopcodes.zMRE END;
Notify: PROCEDURE [POINTER TO CONDITION] =
MACHINE CODE BEGIN Mopcodes.zNOTIFY END;
Broadcast: PROCEDURE [POINTER TO CONDITION] =
MACHINE CODE BEGIN Mopcodes.zBCAST END;
Requeue: PROCEDURE [from: QueueHandle, to: QueueHandle, p: ProcessHandle] =
MACHINE CODE BEGIN Mopcodes.zREQUEUE END;
-- Note: this depends on having one instruction after enabling:
EnableAndRequeue: PRIVATE PROCEDURE [QueueHandle, QueueHandle, ProcessHandle] =
MACHINE CODE BEGIN Mopcodes.zDWDC; Mopcodes.zREQUEUE END;

DisableInterrupts: PROCEDURE = MACHINE CODE BEGIN Mopcodes.zIWDC END;
EnableInterrupts: PROCEDURE = MACHINE CODE BEGIN Mopcodes.zDWDC END;

END.