Solution 2

1.

```
interrupt void intserv() {

  unsigned char buffer, stat, CTCON_saved, CTSTAT_saved, flag;
  unsigned int CNTM_saved, COUNT_saved;

  buffer = *RBUF;                   /* Read Rx buffer */
  stat = *PSTAT;                    /* Read Port A/B Status Register */

  if ((stat & 0x2) == 0) {          /* Port A is not ready */
    CTCON_saved = *CTCON;           /* Save CTCON contents */
    *CTCON = 0x2;                   /* Stop countdown (if running) */
    CTSTAT_saved = *CTSTAT;         /* Save CTSTAT contents */
    COUNT_saved = *COUNT;           /* Save COUNT contents */
    CNTM_saved = *CNTM;             /* Save CNTM contents */
    flag = 0;                       /* Clear timeout indicator */
    *CNTM = 100000;                 /* 0.001-second timeout */
    *CTCON = 0x1;                   /* Start countdown */
    while ((*PSTAT & 0x2) == 0) {   /* While Port A is not ready… */
      if ((*CTSTAT & 0x1) == 1) {
        flag = 1;                   /* Flag a 0.001-second timeout */
        break;                      /* Terminate loop after timeout */
      }
    }
    *CTCON = 0x2;                   /* Stop countdown */
    *CNTM = CNTM_saved;             /* Restore saved CNTM contents */
    *COUNT = COUNT_saved;           /* Restore saved COUNT contents */
    *CTSTAT = CTSTAT_saved;         /* Restore saved CTSTAT contents */
    *CTCON = CTCON_saved;           /* Restore saved CTCON contents */
    if (flag == 0) *PAOUT = buffer; /* OK to output to Port A */
  }

  else *PAOUT = buffer;             /* (stat & 0x2) != 0: Ready */

}
```

2.

The simplest way to check for ***mbuffer*** being empty is to introduce a new variable, say ***items***, keeping track of the number of items in the circular buffer. We still allow writing into ***mbuffer*** even if it is full, but we do not allow the value of the ***items*** variable to exceed BSIZE. (i.e., ***items*** saturates at BSIZE).

Polling:

```
int main() {
  unsigned char mbuffer[BSIZE];     /* Define circular buffer */
  int fin = 0;  int fout = 0;       /* Initialize indices */
  int items = 0;                    /* Initialize item counter */
  *PADIR = 0xFF;                    /* Configure Port A as output */
while (1) {
    while ((*SSTAT & 0x1) == 0) {   /* While RBUF is not ready… */
      if ((*PSTAT & 0x2) != 0) {    /* If PAOUT is ready… */
        if (items > 0) {                  /* If buffer is not empty… */
          *PAOUT = mbuffer[fout];         /* Send character to PAOUT */
          if (fout < BSIZE-1) fout++;     /* Update output index */
          else fout = 0;
          items--;                        /* Update item counter */
        }
      }
    }
    mbuffer[fin] = *RBUF;           /* Get character from RBUF */
    if (fin < BSIZE-1) fin++;       /* Update input index */
    else fin = 0;
    if (items < BSIZE) items++;     /* Update item counter */
  }

  exit(0);
}
```

Interrupt:

```
unsigned char mbuffer[BSIZE];       /* Define circular buffer */
int fin = 0;  int fout = 0;         /* Initialize indices */
int items = 0;                      /* Initialize item counter */

int main() {
  *PADIR = 0xFF;                    /* Configure Port A as output */
  *IVECT = (unsigned int *) &intserv;  /* Set up interrupt vector */
  asm(“MoveControl PSR,#0x40”);        /* CPU responds to IRQ */
  *SCONT = 0x10;                       /* Enable RBUF interrupts */
  *PCONT = 0x20;                       /* Enable PAOUT interrupts */
  while (1);            /* Empty loop, but can code other tasks here */
  exit(0);
}

interrupt void intserv() {
  if ((*SSTAT & 0x10) != 0) {       /* Receiver interrupt flag set? */
    mbuffer[fin] = *RBUF;           /* Get character from RBUF */
    if (fin < BSIZE-1) fin++;       /* Update input index */
    else fin = 0;
    if (items < BSIZE) items++;     /* Update item counter */
  }
  if ((*PSTAT & 0x20) != 0) {       /* IAOUT flag set? */
    if (items > 0) {                /* If buffer is not empty… */
      *PAOUT = mbuffer[fout];       /* Send character to PAOUT */
      if (fout < BSIZE-1) fout++;   /* Update output index */
      else fout = 0;
      items--;                      /* Update item counter */
    }
  }
}
```

3.
Within the reference timeframe of 120, task **T1** is activated 4 times with the deadlines of 30 (t=0, k=0), 60 (t=30, k=1), 90 (t=60, k=2), 120 (t=90, k=3). Therefore, **T1**'s priorities are τ_{10} = 1/30, τ_{11} = 1/60, τ_{12} = 1/90, τ_{13} = 1/120. Task **T2** is activated 3 times with the deadlines of 30 (t=0, k=0), 70 (t=40, k=1), 110 (t=80, k=2). Therefore, **T2**'s priorities are τ_{20} = 1/30, τ_{21} = 1/70, τ_{22} = 1/110. Task **T3** is activated once with the deadline of 120 (t=0, k=0); therefore, **T3**'s priority is τ_{30} = 1/120.

t=0: **T1** [τ_{10}], **T2** [τ_{20}], **T3** [τ_{30}] ready. Both **T1** and **T2** have the highest priority of 1/30. Dispatch **T1** (e.g., because its period is shorter than **T2**'s).

t=10: **T2** [τ_{20}], **T3** [τ_{30}] ready. **T2** has the highest priority of 1/30. Dispatch **T2**.

t=27: **T3** [τ_{30}] ready. Dispatch **T3**.

t=30: **T1** [τ_{11}], **T3** [τ_{30}] (WCET of 7 remains) ready. **T1** has the highest priority of 1/60. Dispatch **T1** (**T3** is suspended).

t=40: **T2** [τ_{21}], **T3** [τ_{30}] (WCET of 7 remains) ready. **T2** has the highest priority of 1/70. Dispatch **T2** (**T3** is suspended).

t=57: **T3** [τ_{30}] (WCET of 7 remains) ready. Dispatch **T3**.

t=60: **T1** [τ_{12}], **T3** [τ_{30}] (WCET of 4 remains) ready. **T1** has the highest priority of 1/90. Dispatch **T1** (**T3** is suspended).

t=70: **T3** [τ_{30}] (WCET of 4 remains) ready. Dispatch **T3**.

t=74: Idle – no tasks to execute.

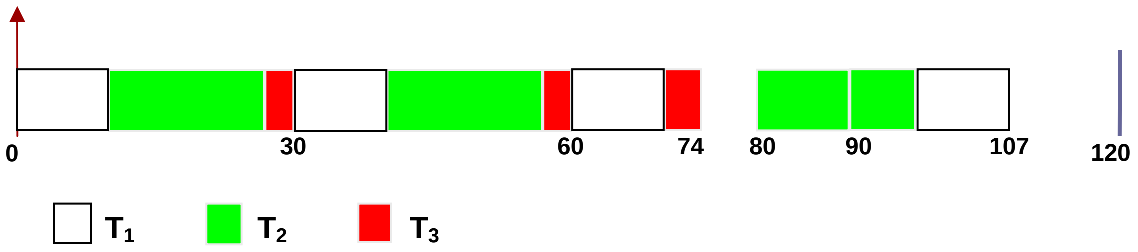
t=80: **T2** [τ_{22}] ready. Dispatch **T2**.

t=90: **T1** [τ_{13}], **T2** [τ_{22}] (WCET of 7 remains) ready. **T2** has the highest priority of 1/110. Dispatch **T2**.

t=97: **T1** [τ_{13}] ready. Dispatch **T1**.

t=107: Idle – no tasks to execute.

t=120: End.


0
30
60
74
80
90
107
120
□ T1
🟩 T2
🟥 T3