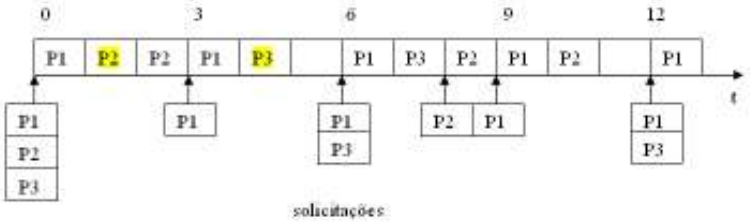
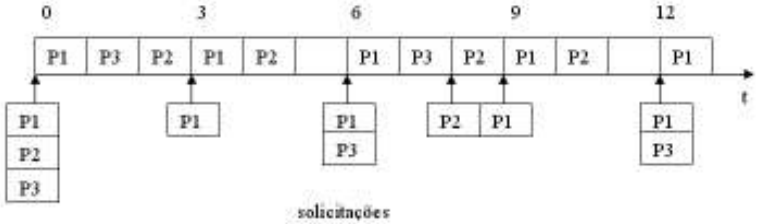
ERRATA
Programação Concorrente em Ambiente Windows – Uma visão de Automação

Capítulo	Página	Versão com erro	Versão Corrigida
3	90	Thread T2 { loop { SeçãoNãoCrítica; c2 = 0; // T1 quer entrar na seção crítica while (c1 != 1) // protocolo de entrada: espera T1 liberar if (vez == 2) { // detecta a contenção c2 = 1; // desiste da tentativa while (vez != 2) ; // espera sua vez de insistir caso empate c2 = 0; // ocupa C1 } // if } // end_loop *SeçãoCrítica;* c2 = 1; // protocolo de Saída: libera T1 vez =1; // é a vez de T1 insistir em caso de empate } // ThreadT2	Thread T2 { loop { SeçãoNãoCrítica; c2 = 0; // T1 quer entrar na seção crítica while (c1 != 1) // protocolo de entrada: espera T1 liberar if (vez == 1) { // detecta a contenção c2 = 1; // desiste da tentativa while (vez != 2) ; // espera sua vez de insistir caso empate c2 = 0; // ocupa C1 } // if } // end_loop *SeçãoCrítica;* c2 = 1; // protocolo de Saída: libera T1 vez =1; // é a vez de T1 insistir em caso de empate } // ThreadT2
3	114	int N1, N2 = 0; Thread T1; { loop { a1: SeçãoNãoCrítica; b1: N1 = 1; // quero acessar a seção crítica c1: N1 = N2 + 1; // pego número da senha d1: while (N2 != 0 && N1 > N2) ; // espero minha vez e1: SeçãoCrítica; f1:N1 = 0; // libero senha } end loop; } end T1; Thread T2;	int N1, N2 = 0; Thread T1; { loop { a1: SeçãoNãoCrítica; b1: N1 = 1; // quero acessar a seção crítica c1: N1 = N2 + 1; // pego número da senha d1: while (N2 != 0 && N1 > N2) ; // espero minha vez e1: SeçãoCrítica; f1: N1 = 0; // libero senha } //end loop; } //end T1; Thread T2;

		{ loop { a2: SeçãoNãoCrítica; b2: N2 = 1; // quero acessar a seção crítica c2: N2 = N1 + 1; // pego número da senha d2: while (N1 != 0 && N2 >= N1) ; // espero minha vez e2: SeçãoCrítica; f2: N1 = 0; // libero senha } end loop; } end T2;	{ loop { a2: SeçãoNãoCrítica; b2: N2 = 1; // quero acessar a seção crítica c2: N2 = N1 + 1; // pego número da senha d2: while (N1 != 0 && N2 >= N1) ; // espero minha vez e2: SeçãoCrítica; f2: N2 = 0; // libero senha } //end loop; } //end T2
4	104	HANDLE ReleaseMutex(	BOOL ReleaseMute(
4	138	HANDLE ReleaseSemaphore(	BOOL ReleaseSemaphore(
4	143	HANDLE SetEvent(HANDLE hEvent); Define o estado do evento como sinalizado. HANDLE ResetEvent(HANDLE hEvent); Define o estado do evento como não sinalizado. HANDLE PulseEvent(HANDLE hEvent);	BOOL SetEvent(HANDLE hEvent); Define o estado do evento como sinalizado. BOOL ResetEvent(HANDLE hEvent); Define o estado do evento como não sinalizado. BOOL PulseEvent(HANDLE hEvent);
4	152	HANDLE SetWaitableTimer(	BOOL SetWaitableTimer(

4	169	Um evento com reset manual ficará sinalizado se a função *SetEvent()* for chamada, e só passará ao estado não sinalizado após chamada da função *RestEvent().*	Um evento com reset manual ficará sinalizado se a função *SetEvent()* for chamada, e só passará ao estado não sinalizado após chamada da função *ResetEvent().*
7	304	switch (pagina) { case 0: // Página 0 if (Lflag) Leitura(); if (Cflag) Controle(); if (Sflag) Saída(); break; **********	switch (pagina) { case 0: // Página 0 if (Lflag) Entrada(); if (Cflag) Controle(); if (Sflag) Saída(); break; **********
7	308	O escalonador fará Pri_1 > Pri_2 > Pri_3  **Figura 7.8** Escalonamento para algoritmo RM	O escalonador fará Pri_1 > Pri_3 > Pri_2  **Figura 7.8** Escalonamento para algoritmo RM
7	335	Consta da criação repetida de uma thread por milhares de ciclos. O tempo médio de criação foi de 100µs. O tempo máximo foi de 1µs, verificado em apenas 0.01 % dos casos (2 casos).	Consta da criação repetida de uma thread por milhares de ciclos. O tempo médio de criação foi de 100µs. O tempo máximo foi de 1ms, verificado em apenas 0.01 % dos casos (2 casos).