```
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <Windows.h>
#include "parking2.h"

/*************************/
/*   Headers manuales     */
/*************************/

#define ROAD_LENGTH               80
#define EMPTY                      0
#define RESERVED                  1

#define LOTS                      0
#define STRIPE                     1
#define LANE                      2

#define N_AJUSTES                  4

#define TEN_THOUSAND            10000

/* STRUCTS DE ARGUMENTOS DE HILO */
struct PARKING_inicio_args {
    TIPO_FUNCION_LLEGADA *aFuncionesLlegada;
    TIPO_FUNCION_SALIDA *aFuncionesSalida;
    long intervalo;
    int d;
};

/* Punteros a las funciones */
int(*PARKING_inicio)(TIPO_FUNCION_LLEGADA *, TIPO_FUNCION_SALIDA *, long, int);
int(*PARKING_fin)(void);
int(*PARKING_aparcar)(HCoche, void *datos, TIPO_FUNCION_APARCAR_COMMIT,
    TIPO_FUNCION_PERMISO_AVANCE,
    TIPO_FUNCION_PERMISO_AVANCE_COMMIT);
int(*PARKING_desaparcar)(HCoche, void *datos,
    TIPO_FUNCION_PERMISO_AVANCE,
    TIPO_FUNCION_PERMISO_AVANCE_COMMIT);
int(*PARKING_getNUmero)(HCoche);
int(*PARKING_getLongitud)(HCoche);
int(*PARKING_getPosiciOnEnAcera)(HCoche);
unsigned long(*PARKING_getTServ)(HCoche);
int(*PARKING_getColor)(HCoche);
void * (*PARKING_getDatos)(HCoche);
int(*PARKING_getX)(HCoche);
int(*PARKING_getY)(HCoche);
int(*PARKING_getX2)(HCoche);
int(*PARKING_getY2)(HCoche);
int(*PARKING_getAlgoritmo)(HCoche);
int(*PARKING_isAceraOcupada)(int algoritmo, int pos);

/* Funciones de llegada */
int LlegadaPrimerAjuste(HCoche hc);
int LlegadaSiguienteAjuste(HCoche hc);
int LlegadaMejorAjuste(HCoche hc);
int LlegadaPeorAjuste(HCoche hc);

int SalidaPrimerAjuste(HCoche hc);
int SalidaSiguienteAjuste(HCoche hc);
int SalidaMejorAjuste(HCoche hc);
int SalidaPeorAjuste(HCoche hc);

int SiguienteAjustePrimerHueco(int last_pos);
int SiguienteAjusteAjustaPrimer(int c_length, int a, int b);

/* Otras funciones */
int TestArgs(int argc, char * argv[]);
int InvalidOptionMsg(char * opt);
int HelpMsg(void);
int ShortHelpMsg(void);
int LoadParkingDll();
void AparcarCommit(HCoche hc);
void PermisoAvance(HCoche hc);
void PermisoAvanceCommit(HCoche hc);
int CreateIPC();
DWORD WINAPI HiloParkingInicio(LPVOID param);
DWORD WINAPI HiloAparcar(LPVOID param);
DWORD WINAPI HiloDesaparcar(LPVOID param);
int CloseIPC();
void printCarretera(); // __DEBUG__ ONLY


// Descomentar líneas para desactivar ajustes
//#define PRIMER_AJUSTE_OFF
//#define SIGUIENTE_AJUSTE_OFF
//#define MEJOR_AJUSTE_OFF
//#define PEOR_AJUSTE_OFF

#define ARRAY_SYNC // Comentar esta línea si se desea sincronizacion con bucle espera (no ocupada)
#if !defined(LOOPING_SYNC) && !defined(ARRAY_SYNC)
#define LOOPING_SYNC
#endif

struct IPC {
    int nLotsArray[N_AJUSTES][ROAD_LENGTH];                    // Acera
    HANDLE hRoadSemaphores[N_AJUSTES][ROAD_LENGTH];            // Carretera

#ifdef ARRAY_SYNC
    HANDLE aTenThousandSemaphoresArray[N_AJUSTES][TEN_THOUSAND];
#elif // LOOPING_SYNC
    HANDLE hMutexOrdenCoches[N_AJUSTES];
    HANDLE hEventOrdenCoches[N_AJUSTES];
    int nNextCar[N_AJUSTES];
#endif

    HINSTANCE controladoraDLL;
};

struct Car {
    HCoche hCocheSM;
```

```
112 };
113
114 struct IPC IpcResources;
115 struct Car SharedMemory;
116
117 int main(int argc, char *argv[])
118 {
119     int nIntervalo, nDebugFlag;
120     struct PARKING_inicio_args piargs;
121
122     TIPO_FUNCION_LLEGADA aFuncionesLlegada[N_AJUSTES] = {
123         LlegadaPrimerAjuste,
124         LlegadaSiguienteAjuste,
125         LlegadaMejorAjuste,
126         LlegadaPeorAjuste
127     };
128
129     TIPO_FUNCION_SALIDA aFuncionesSalida[N_AJUSTES] = {
130         SalidaPrimerAjuste,
131         SalidaSiguienteAjuste,
132         SalidaMejorAjuste,
133         SalidaPeorAjuste
134     };
135
136     if (-1 == LoadParkingDll()) {
137         fprintf(stderr, "%s\n", "Main: LoadParkingDll");
138     }
139
140     /* Creación de Carretera de MUTEXES*/
141     if (CreateIPC() == -1) {
142         fprintf(stderr, "%s\n", "Main: CreateIPC");
143         return -1;
144     }
145
146     nDebugFlag = TestArgs(argc, argv);
147     if (-1 == nDebugFlag)
148         return -1;
149     else
150         nIntervalo = atoi(argv[1]); // Primer argumento, velocidad
151
152     piargs.aFuncionesLlegada = aFuncionesLlegada;
153     piargs.aFuncionesSalida = aFuncionesSalida;
154     piargs.intervalo = nIntervalo;
155     piargs.d = nDebugFlag;
156
157
158     if (NULL == CreateThread(NULL, 0, HiloParkingInicio, (void *)&piargs, 0, NULL)) {
159         PERROR("main: CreateThread: HiloParkingInicio");
160         return -1;
161     }
162
163     Sleep(30000);
164
165     if (-1 == PARKING_fin()) {
166         fprintf(stderr, "%s\n", "PARKING_fin: finalizado con error");
167         return -1;
168     }
169
170     if (CloseIPC() == -1) {
171         fprintf(stderr, "%s\n", "Main: CloseIPC");
172         return -1;
173     }
174
175     FreeLibrary(IpcResources.controladoraDLL);
176
177     return 0;
178 }
179
180 int LoadParkingDll()
181 {
182     HINSTANCE controladorDll;
183
184     controladorDll = LoadLibrary("parking2.dll");
185     if (NULL == controladorDll) {
186         fprintf(stderr, "%s\n", "ErrorLoadLibrary");
187         return -1;
188     }
189
190     IpcResources.controladoraDLL = controladorDll;
191
192     PARKING_inicio = (int(*)(TIPO_FUNCION_LLEGADA *, TIPO_FUNCION_SALIDA *, LONG, BOOL)) GetProcAddress(controladorDll, "PARKING2_inicio");
193     if (PARKING_inicio == NULL) {
194         PERROR("LoadParkingDll: PARKING_inicio");
195         return -1;
196     }
197
198     PARKING_fin = (int(*)(void)) GetProcAddress(controladorDll, "PARKING2_fin");
199     if (PARKING_fin == NULL) {
200         PERROR("LoadParkingDll: PARKING_fin");
201         return -1;
202     }
203
204     PARKING_aparcar = (int(*)(HCoche, void *datos, TIPO_FUNCION_APARCAR_COMMIT, TIPO_FUNCION_PERMISO_AVANCE, TIPO_FUNCION_PERMISO_AVANCE_COMMIT)) GetProcAddress(
205     if (PARKING_aparcar == NULL) {
206         PERROR("LoadParkingDll: PARKING_aparcar");
207         return -1;
208     }
209
210     PARKING_desaparcar = (int(*)(HCoche, void *datos, TIPO_FUNCION_PERMISO_AVANCE, TIPO_FUNCION_PERMISO_AVANCE_COMMIT)) GetProcAddress(controladorDll, "PARKING2_
211     if (PARKING_desaparcar == NULL) {
212         PERROR("LoadParkingDll: PARKING_desaparcar");
213         return -1;
214     }
215
216     PARKING_getNUmero = (int(*)(HCoche)) GetProcAddress(controladorDll, "PARKING2_getNUmero");
217     if (PARKING_getNUmero == NULL) {
218         PERROR("LoadParkingDll: PARKING_getNUmero");
219         return -1;
220     }
221
222     PARKING_getLongitud = (int(*)(HCoche)) GetProcAddress(controladorDll, "PARKING2_getLongitud");
223     if (PARKING_getLongitud == NULL) {
```

```
224         PERROR("LoadParkingDll: PARKING_getLongitud");
225         return -1;
226     }
227 
228     PARKING_getPosiciOnEnAcera = (int(*)(HCoche)) GetProcAddress(controladorDll, "PARKING2_getPosiciOnEnAcera");
229     if (PARKING_getPosiciOnEnAcera == NULL) {
230         PERROR("LoadParkingDll: PARKING_getPosiciOnEnAcera");
231         return -1;
232     }
233 
234     PARKING_getTServ = (unsigned long(*)(HCoche)) GetProcAddress(controladorDll, "PARKING2_getTServ");
235     if (PARKING_getTServ == NULL) {
236         PERROR("LoadParkingDll: PARKING_getTServ");
237         return -1;
238     }
239 
240     PARKING_getColor = (int(*)(HCoche)) GetProcAddress(controladorDll, "PARKING2_getColor");
241     if (PARKING_getColor == NULL) {
242         PERROR("LoadParkingDll: PARKING_getColor");
243         return -1;
244     }
245 
246     PARKING_getDatos = (void * (*)(HCoche)) GetProcAddress(controladorDll, "PARKING2_getDatos");
247     if (PARKING_getDatos == NULL) {
248         PERROR("LoadParkingDll: PARKING_getDatos");
249         return -1;
250     }
251 
252     PARKING_getX = (int(*)(HCoche)) GetProcAddress(controladorDll, "PARKING2_getX");
253     if (PARKING_getX == NULL) {
254         PERROR("LoadParkingDll: PARKING_getX");
255         return -1;
256     }
257 
258     PARKING_getY = (int(*)(HCoche)) GetProcAddress(controladorDll, "PARKING2_getY");
259     if (PARKING_getY == NULL) {
260         PERROR("LoadParkingDll: PARKING_getY");
261         return -1;
262     }
263 
264     PARKING_getX2 = (int(*)(HCoche)) GetProcAddress(controladorDll, "PARKING2_getX2");
265     if (PARKING_getX2 == NULL) {
266         PERROR("LoadParkingDll: PARKING_getX2");
267         return -1;
268     }
269 
270     PARKING_getY2 = (int(*)(HCoche)) GetProcAddress(controladorDll, "PARKING2_getY2");
271     if (PARKING_getY2 == NULL) {
272         PERROR("LoadParkingDll: PARKING_getY2");
273         return -1;
274     }
275 
276     PARKING_getAlgoritmo = (int(*)(HCoche)) GetProcAddress(controladorDll, "PARKING2_getAlgoritmo");;
277     if (PARKING_getAlgoritmo == NULL) {
278         PERROR("LoadParkingDll: PARKING_getAlgoritmo");
279         return -1;
280     }
281 
282     PARKING_isAceraOcupada = (int(*)(int algoritmo, int pos)) GetProcAddress(controladorDll, "PARKING2_isAceraOcupada");
283     if (PARKING_isAceraOcupada == NULL) {
284         PERROR("LoadParkingDll: PARKING_isAceraOcupada");
285         return -1;
286     }
287 
288     return 0;
289 }
290 
291 int CreateIPC(void)
292 {
293     int i, j;
294     HANDLE hAuxHandle;
295 
296     /* Creacion carretera de mutex*/
297     for (i = 0; i < N_AJUSTES; ++i) {
298         for (j = 0; j < ROAD_LENGTH; ++j) {
299             hAuxHandle = CreateSemaphore(NULL, 1, 1, NULL);
300             if (hAuxHandle == NULL) {
301                 PERROR("CreateIPC: CreateSemaphore: 1");
302                 return -1;
303             }
304             IpcResources.hRoadSemaphores[i][j] = hAuxHandle;
305         }
306     }
307 
308 
309     /* Inicialización de las aceras */
310     for (i = 0; i < N_AJUSTES; ++i) {
311         for (j = 0; j < ROAD_LENGTH; ++j) {
312             IpcResources.nLotsArray[i][j] = EMPTY;
313         }
314     }
315 
316 #ifdef ARRAY_SYNC
317     /* Creacion array bidimensional de semaforos para controlar el orden de los coches*/
318     for (j = 0; j < N_AJUSTES; j++) {
319 
320         hAuxHandle = CreateSemaphore(NULL, 1, 1, NULL);
321 
322         if (hAuxHandle == NULL) {
323             PERROR("CreateIPC: CreateSemaphore: 2_1");
324             return -1;
325         }
326 
327         IpcResources.aTenThousandSemaphoresArray[j][0] = hAuxHandle;
328 
329         for (i = 1; i < TEN_THOUSAND; ++i) {
330             hAuxHandle = CreateSemaphore(NULL, 0, 1, NULL);
331             if (hAuxHandle == NULL) {
332                 PERROR("CreateIPC: CreateSemaphore: 2_2");
333                 return -1;
334             }
335 
```

```
336             IpcResources.aTenThousandSemaphoresArray[j][i] = hAuxHandle;
337         }
338     }
339
340 #elif //LOOPING_SYNC
341
342     /* Creación Mutex Ordenación Coches */
343     for (i = 0; i < N_AJUSTES; i++){
344         IpcResources.hMutexOrdenCoches[i] = CreateMutex(NULL, 0, NULL);
345         if (IpcResources.hMutexOrdenCoches == NULL) {
346             PERROR("CreateIPC: CreateMutex: hMutexOrdenCoches");
347             return -1;
348         }
349     }
350
351
352     /* Creación Evento Orden Coches */
353     for (i = 0; i < N_AJUSTES; i++) {
354         IpcResources.hEventOrdenCoches[i] = CreateEvent(NULL, 1, 1, NULL);
355         if (IpcResources.hEventOrdenCoches == NULL) {
356             PERROR("CreateIPC: CreateEvent: hEventOrdenCoches");
357             return -1;
358         }
359     }
360
361     for (i = 0; i < N_AJUSTES; i++) {
362         IpcResources.nNextCar[i] = 1;
363     }
364
365 #endif
366
367     return 0;
368 }
369
370 int CloseIPC(void) {
371     int i, j;
372
373     /* Cierre de los handle de la carretera de mutex*/
374     for (i = 0; i < N_AJUSTES; ++i) {
375         for (j = 0; j < ROAD_LENGTH; ++j) {
376             if (CloseHandle(IpcResources.hRoadSemaphores[i][j]) == 0) {
377                 PERROR("CloseIPC: CloseHandle: hRoadSemaphores");
378                 return -1;
379             }
380         }
381     }
382
383 #ifdef ARRAY_SYNC
384     /* Creacion array bidimensional de semaforos para controlar el orden de los coches*/
385     for (j = 0; j < N_AJUSTES; j++) {
386         for (i = 0; i < TEN_THOUSAND; ++i) {
387             if (CloseHandle(IpcResources.aTenThousandSemaphoresArray[j][i]) == 0) {
388                 PERROR("CloseIPC: CloseHandle: aTenThousandSemaphoresArray");
389                 return -1;
390             }
391         }
392     }
393 #endif
394     return 0;
395 }
396
397 int TestArgs(int argc, char * argv[])
398 {
399     int i,  debug_flag = 0;
400     size_t j, length;
401
402     /* Can receive between 2 and 3 arguments. Remember that argv[0] is file name
403     * so limits are 3 and 4
404     */
405
406     switch (argc) {
407
408     case 3:
409         if (strcmp(argv[2], "D") && strcmp(argv[2], "d")) {
410             InvalidOptionMsg(argv[2]);
411             return -1;
412         }
413         else {
414             debug_flag = 1;
415             length = strlen(argv[1]);
416             for (j = 0; j < length; ++j) {
417                 if (!isdigit(argv[1][j])) {
418                     InvalidOptionMsg(argv[1]);
419                     return -1;
420                 }
421             }
422             break;
423         }
424
425     case 2:
426         if (!strcmp(argv[1], "--help")) {
427             HelpMsg();
428             return -1;
429         }
430
431         length = strlen(argv[1]);
432         for (j = 0; j < length; ++j) {
433             if (!isdigit(argv[1][j])) {
434                 InvalidOptionMsg(argv[1]);
435                 return -1;
436             }
437         }
438         break;
439
440     default:
441         ShortHelpMsg();
442         return -1;
443     }
444
445     return debug_flag;
446 }
447
```

```
448 int InvalidOptionMsg(char * opt)
449 {
450     char msg[] = "parking: invalid option -- '%s'\n\
451 Try 'parking --help' for more information.\n";
452 
453     return printf(msg, opt);
454 }
455 
456 int HelpMsg(void)
457 {
458 
459     char msg[] = "Usage: parking SPEED [D]\n\
460 Simulates a process  allocation  system with  an  interface  that  emulates a\n\
461 road with cars which must be parked in a ordered fashion.\n\
462     SPEED               Controls the speed of events. 0 is the  higher speed,\n\
463                         and subsequent  incresing values, up to INT_MAX, slow\n\
464                         down the simulation.\n\
465     D                   A 'D' can be used  to  produce debugging  information\n\
466                         about the simulation, which will be output on stderr.";
467 
468     return printf("%s\n", msg);
469 
470 }
471 
472 int ShortHelpMsg(void)
473 {
474 
475     char msg[1000] = "Usage: parking SPEED [D]\n\
476 Try 'parking --help' for more information.\n";
477 
478     return printf("%s", msg);
479 
480 }
481 
482 int LlegadaPrimerAjuste(HCoche hc)
483 {
484 #ifdef PRIMER_AJUSTE_OFF //__DEBUG__
485     return -2;
486 #else
487 
488     int c_length;
489     int i, j, pos;
490 
491     c_length = PARKING_getLongitud(hc);
492 
493     pos = -1;
494     for (i = 0; i <= ROAD_LENGTH - c_length; ++i) {
495 
496         if (IpcResources.nLotsArray[PRIMER_AJUSTE][i] == EMPTY) {
497 
498             j = i;
499             while (j < i + c_length) {
500                 if (EMPTY == IpcResources.nLotsArray[PRIMER_AJUSTE][j])
501                     ++j;
502                 else
503                     break;
504             }
505 
506             if (j == i + c_length) {
507 
508                 for (j = i; j < i + c_length; j++) {
509                     IpcResources.nLotsArray[PRIMER_AJUSTE][j] = RESERVED;
510                 }
511 
512                 pos =  i;
513                 break;
514             }
515         }
516     }
517 
518     if (pos != -1) {
519 
520         if (NULL == CreateThread(NULL, 0, HiloAparcar, (LPVOID)hc, 0, NULL)) {
521             PERROR("LlegadaPrimerAjuste: CreateThread: HiloAparcar");
522             return -2;
523         }
524     }
525 
526     return pos;
527 
528 #endif
529 }
530 
531 int LlegadaSiguienteAjuste(HCoche hc)
532 {
533 #ifdef SIGUIENTE_AJUSTE_OFF //__DEBUG__
534     return -2;
535 #else
536 
537 
538     static int last_pos = -1;
539     int c_length;
540     int i, first_spot, free_spot;
541 
542     c_length = PARKING_getLongitud(hc);
543 
544     /* Se podría hacer con el módulo de la posición last_pos % ROAD_LENGTH,
545     * pero creo que queda más complejo por el tratamiento especial al valor
546     * de last_pos 0. Por ello, corrijo manualmente cuando last_pos se pasa
547     * de ROAD_LENGTH -1
548     */
549 
550     if (last_pos > -1) {
551         if (EMPTY != IpcResources.nLotsArray[SIGUIENTE_AJUSTE][last_pos]) {
552             first_spot = SiguienteAjustePrimerHueco(last_pos);
553             if (-1 == first_spot) {
554                 return -1;
555             }
556         }
557         else {
558             first_spot = last_pos;
559             while (IpcResources.nLotsArray[SIGUIENTE_AJUSTE][first_spot] == EMPTY && first_spot>0) first_spot--;
```

```
560         }
561     }
562     else {
563         first_spot = 0;
564     }
565 
566     free_spot = SiguienteAjusteAjustaPrimer(c_length,
567         first_spot,
568         ROAD_LENGTH);
569 
570     if (-1 == free_spot) {
571         free_spot = SiguienteAjusteAjustaPrimer(c_length, 0,
572             first_spot);
573     }
574 
575     if (free_spot != -1) {
576 
577         for (i = free_spot; i < free_spot + c_length; ++i) {
578             IpcResources.nLotsArray[SIGUIENTE_AJUSTE][i] = RESERVED;
579         }
580 
581         last_pos = free_spot;
582 
583         if (NULL == CreateThread(NULL, 0, HiloAparcar, (LPVOID)hc, 0, NULL)) {
584             PERROR("LlegadaPrimerAjuste: CreateThread: HiloAparcar");
585             return -2;
586         }
587     }
588 
589     return free_spot;
590 
591 #endif
592 
593 }
594 
595 int SiguienteAjustePrimerHueco(int last_pos)
596 {
597     int i;
598 
599     for (i = last_pos + 1; i < ROAD_LENGTH; ++i) {
600         if (EMPTY == IpcResources.nLotsArray[SIGUIENTE_AJUSTE][i])
601             return i;
602     }
603 
604     for (i = 0; i < last_pos; ++i) {
605         if (EMPTY == IpcResources.nLotsArray[SIGUIENTE_AJUSTE][i])
606             return i;
607     }
608 
609     return -1;
610 }
611 
612 int SiguienteAjusteAjustaPrimer(int c_length, int a, int b)
613 {
614     int i, j;
615 
616     for (i = a; i <= b - c_length; ++i) {
617 
618         if (IpcResources.nLotsArray[SIGUIENTE_AJUSTE][i] == EMPTY) {
619 
620             for (j = i; j < i + c_length; j++) {
621                 if (IpcResources.nLotsArray[SIGUIENTE_AJUSTE][j] != EMPTY)
622                     break;
623             }
624 
625             if (j == i + c_length) {
626                 return i;
627             }
628         }
629     }
630 
631     return -1;
632 }
633 
634 int LlegadaMejorAjuste(HCoche hc)
635 {
636 #ifdef MEJOR_AJUSTE_OFF //__DEBUG__
637     return -2;
638 #else
639 
640     int i, j;
641     int c_length;
642     int size, n_huecos, bestfit;
643     int huecos[ROAD_LENGTH] = { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
644         0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
645         0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
646         0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
647     };
648 
649     c_length = PARKING_getLongitud(hc);
650 
651     /* Buscamos los huecos en que el coche cabe */
652     n_huecos = 0;
653     i = 0;
654     while (i <= ROAD_LENGTH - c_length) {
655 
656         size = 0;
657 
658         if (EMPTY == IpcResources.nLotsArray[MEJOR_AJUSTE][i]) {
659 
660             for (j = i; j < ROAD_LENGTH
661                 && EMPTY == IpcResources.nLotsArray[MEJOR_AJUSTE][j]; ++j) {
662                 ++size;
663             }
664 
665             if (size >= c_length) {
666                 huecos[i] = size;
667                 ++n_huecos;
668                 i += size;
669             }
670             else {
671                 ++i;
```

```
672             }
673 
674 }
675         else {
676             ++i;
677         }
678     }
679 
680     if (n_huecos < 1) {
681         return -1;
682     }
683 
684     /* Seleccionamos el hueco más pequeño */
685     for (i = 0; i < ROAD_LENGTH; ++i) {
686         if (huecos[i] != 0) {
687             bestfit = i;
688             break;
689         }
690     }
691 
692     for (i = bestfit + 1; i < ROAD_LENGTH; ++i) {
693         if (huecos[i] != 0 && huecos[i] < huecos[bestfit]) {
694             bestfit = i;
695         }
696     }
697 
698     for (i = bestfit; i < bestfit + c_length; ++i)
699         IpcResources.nLotsArray[MEJOR_AJUSTE][i] = RESERVED;
700 
701     if (NULL == CreateThread(NULL, 0, HiloAparcar, (LPVOID)hc, 0, NULL)) {
702         PERROR("LlegadaPrimerAjuste: CreateThread: HiloAparcar");
703         return -2;
704     }
705 
706 
707     return bestfit;
708 
709 
710 #endif
711 
712     // Crear hilo de aparcar
713     // Devolver posición
714 }
715 
716 int LlegadaPeorAjuste(HCoche hc)
717 {
718 #ifdef PEOR_AJUSTE_OFF //__DEBUG__
719     return -2;
720 #else
721 
722     int i, j;
723     int c_length;
724     int size, n_huecos, worstfit;
725     int huecos[ROAD_LENGTH] = { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
726         0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
727         0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
728         0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
729     };
730 
731     c_length = PARKING_getLongitud(hc);
732 
733     /* Buscamos los huecos en que el coche cabe */
734     n_huecos = 0;
735     i = 0;
736     while (i <= ROAD_LENGTH - c_length) {
737 
738         size = 0;
739 
740         if (EMPTY == IpcResources.nLotsArray[PEOR_AJUSTE][i]) {
741 
742             for (j = i; j < ROAD_LENGTH
743                 && EMPTY == IpcResources.nLotsArray[PEOR_AJUSTE][j]; ++j) {
744                 ++size;
745             }
746 
747             if (size >= c_length) {
748                 huecos[i] = size;
749                 ++n_huecos;
750                 i += size;
751             }
752             else {
753                 ++i;
754             }
755 
756         }
757         else {
758             ++i;
759         }
760     }
761 
762     if (n_huecos < 1) {
763         return -1;
764     }
765 
766     /* Seleccionamos el hueco más grande */
767     for (i = 0; i < ROAD_LENGTH; ++i) {
768         if (huecos[i] != 0) {
769             worstfit = i;
770             break;
771         }
772     }
773 
774     for (i = worstfit + 1; i < ROAD_LENGTH; ++i) {
775         if (huecos[i] != 0 && huecos[i] > huecos[worstfit]) {
776             worstfit = i;
777         }
778     }
779 
780     for (i = worstfit; i < worstfit + c_length; ++i)
781         IpcResources.nLotsArray[PEOR_AJUSTE][i] = RESERVED;
782 
783     if (NULL == CreateThread(NULL, 0, HiloAparcar, (LPVOID)hc, 0, NULL)) {
```

```
784         PERROR("LlegadaPrimerAjuste: CreateThread: HiloAparcar");
785         return -2;
786     }
787 
788     return worstfit;
789 
790 #endif
791 
792     // Crear hilo de aparcar
793     // Devolver posición
794 }
795 
796 int SalidaPrimerAjuste(HCoche hc)
797 {
798 #if 0 //__DEBUG__
799     return -2;
800 #else
801 
802     if (NULL == CreateThread(NULL, 0, HiloDesaparcar, (LPVOID)hc, 0, NULL)) {
803         PERROR("SalidaPrimerAjuste: CreateThread: HiloDesparcar");
804         return -1;
805     }
806 
807     return 0;
808 
809 #endif
810 }
811 
812 int SalidaSiguienteAjuste(HCoche hc)
813 {
814 #ifdef SIGUIENTE_AJUSTE_OFF //__DEBUG__
815     return -2;
816 #else
817 
818     if (NULL == CreateThread(NULL, 0, HiloDesaparcar, (LPVOID)hc, 0, NULL)) {
819         PERROR("PrimerAjuste: CreateThread: HiloAparcar");
820         return -1;
821     }
822 
823     return 0;
824 
825 #endif
826 }
827 
828 int SalidaMejorAjuste(HCoche hc)
829 {
830 #ifdef MEJOR_AJUSTE_OFF //__DEBUG__
831     return -2;
832 #else
833 
834     if (NULL == CreateThread(NULL, 0, HiloDesaparcar, (LPVOID)hc, 0, NULL)) {
835         PERROR("PrimerAjuste: CreateThread: HiloAparcar");
836         return -1;
837     }
838 
839     return 0;
840 
841 #endif
842 }
843 
844 int SalidaPeorAjuste(HCoche hc)
845 {
846 #ifdef PEOR_AJUSTE_OFF //__DEBUG__
847     return -2;
848 #else
849 
850     if (NULL == CreateThread(NULL, 0, HiloDesaparcar, (LPVOID)hc, 0, NULL)) {
851         PERROR("PrimerAjuste: CreateThread: HiloAparcar");
852         return -1;
853     }
854 
855     return 0;
856 
857 #endif
858 }
859 
860 DWORD WINAPI HiloParkingInicio(LPVOID param)
861 {
862     struct PARKING_inicio_args args;
863     args = *(struct PARKING_inicio_args *) param;
864     PARKING_inicio(args.aFuncionesLlegada, args.aFuncionesSalida, args.intervalo, args.d);
865     return 0;
866 }
867 
868 DWORD WINAPI HiloAparcar(LPVOID param)
869 {
870     /* ORDEN DE LOS COCHES  */
871     HCoche hc;
872     int nNumeroCoche;
873     int algoritmo;
874 
875     hc = (HCoche) param;
876     nNumeroCoche = PARKING_getNUmero(hc);
877     algoritmo = PARKING_getAlgoritmo(hc);
878 
879 #ifdef ARRAY_SYNC
880 
881     if (WaitForSingleObject(
882         IpcResources.aTenThousandSemaphoresArray[algoritmo][(nNumeroCoche - 1) % TEN_THOUSAND], INFINITE
883     ) == WAIT_FAILED) { // Condicion y espera atomicas
884         PERROR("HiloAparcar: WaitForSingleObject");
885     }
886 
887 #else // LOOPING_SYNC
888     if (WaitForSingleObject(IpcResources.hMutexOrdenCoches[algoritmo], INFINITE) == WAIT_FAILED) { // Condicion y espera atomicas
889         PERROR("HiloAparcar: WaitForSingleObject");
890     }
891 
892         while (nNumeroCoche != IpcResources.nNextCar[algoritmo]) {
893 
894             if (SignalObjectAndWait(IpcResources.hMutexOrdenCoches[algoritmo], IpcResources.hEventOrdenCoches[algoritmo], INFINITE, 0) == WAIT_FAILED) {
895                 PERROR("HiloAparcar: WaitForSingleObject");
```

```
896             }
897
898             if (WaitForSingleObject(IpcResources.hMutexOrdenCoches[algoritmo], INFINITE) == WAIT_FAILED) {
899                 PERROR("HiloAparcar: WaitForSingleObject");
900             }
901
902         }
903
904     if (ReleaseMutex(IpcResources.hMutexOrdenCoches[algoritmo]) == 0) {
905         PERROR("HiloAparcar: ReleaseMutex");
906     }
907 #endif
908
909     PARKING_aparcar(hc, NULL, AparcarCommit, PermisoAvance, PermisoAvanceCommit);
910
911     return 0;
912 }
913
914 DWORD WINAPI HiloDesaparcar(LPVOID param)
915 {
916     HCoche hc;
917
918     hc = (HCoche)param;
919
920     PARKING_desaparcar(hc, NULL, PermisoAvance, PermisoAvanceCommit);
921
922     return 0;
923 }
924
925 void AparcarCommit(HCoche hc)
926 {
927     int algoritmo = PARKING_getAlgoritmo(hc);
928     int nNumeroCoche = PARKING_getNUmero(hc);
929
930 #ifdef ARRAY_SYNC
931
932     if (ReleaseSemaphore(IpcResources.aTenThousandSemaphoresArray[algoritmo][nNumeroCoche % TEN_THOUSAND], 1, NULL) == 0){
933         PERROR("AparcarCommit: ReleaseSemaphore");
934     }
935
936 #else
937
938     if (WaitForSingleObject(IpcResources.hMutexOrdenCoches[algoritmo], INFINITE) == WAIT_FAILED) { // Condicion y espera atomicas
939         PERROR("AparcarCommit: WaitForSingleObject");
940     }
941
942     IpcResources.nNextCar[algoritmo] = nNumeroCoche + 1;
943
944     if (PulseEvent(IpcResources.hEventOrdenCoches[algoritmo]) == 0) {
945         PERROR("AparcarCommit: PulseEvent");
946     }
947
948     if (ReleaseMutex(IpcResources.hMutexOrdenCoches[algoritmo]) == 0) {
949         PERROR("AparcarCommit: ReleaseMutex");
950     }
951 #endif
952 }
953
954 void PermisoAvance(HCoche hc)
955 {
956     int i;
957     int X_inicio, X_fin, Y_inicio, Y_fin, algoritmo, c_length;
958
959     X_inicio = PARKING_getX(hc);
960     X_fin = PARKING_getX2(hc);
961     Y_inicio = PARKING_getY(hc);
962     Y_fin = PARKING_getY2(hc);
963     algoritmo = PARKING_getAlgoritmo(hc);
964     c_length = PARKING_getLongitud(hc);
965
966     if (X_fin >= 0
967         && X_fin < ROAD_LENGTH){
968         if (Y_inicio == LANE
969             && Y_fin == LANE) { // Horizontal
970
971             if (WaitForSingleObject(IpcResources.hRoadSemaphores[algoritmo][X_fin], INFINITE) == WAIT_FAILED) {
972                 PERROR("PermisoAvance: WaitForSingleObject 1");
973             }
974         }
975         else if (STRIPE == Y_inicio && LANE == Y_fin) { // Desaparcando
976             for (i = X_fin + c_length - 1; i >= X_fin; --i) {
977                 if (WaitForSingleObject(IpcResources.hRoadSemaphores[algoritmo][i], INFINITE) == WAIT_FAILED) {
978                     PERROR("PermisoAvance: WaitForSingleObject 2");
979                 }
980             }
981
982         }
983     }
984 }
985
986 void PermisoAvanceCommit(HCoche hc)
987 {
988     int i;
989     int X_inicio, X_fin, Y_inicio, Y_fin, algoritmo, c_length;
990
991     X_inicio = PARKING_getX2(hc);
992     X_fin = PARKING_getX(hc);
993     Y_inicio = PARKING_getY2(hc);
994     Y_fin = PARKING_getY(hc);
995     algoritmo = PARKING_getAlgoritmo(hc);
996     c_length = PARKING_getLongitud(hc);
997
998     if (X_inicio + c_length - 1 >= 0
999         && X_inicio + c_length - 1 < ROAD_LENGTH) {
1000
1001         if (LANE == Y_inicio
1002             && LANE == Y_fin) { // Horizontal
1003
1004             if (ReleaseSemaphore(IpcResources.hRoadSemaphores[algoritmo][X_inicio + c_length - 1], 1, NULL) == 0) {
1005                 PERROR("PermisoAvanceCommit: ReleaseSemaphore 1");
1006             }
1007
```

```
1008         }
1009         else if (STRIPE == Y_inicio && LANE == Y_fin) {
1010             for (i = X_inicio + c_length - 1; i >= X_inicio; --i){
1011                 IpcResources.nLotsArray[algoritmo][i] = EMPTY;
1012             }
1013         }
1014         else if (LANE == Y_inicio && STRIPE == Y_fin) {
1015             for (i = X_inicio + c_length - 1; i >= X_inicio; --i) {
1016                 if (ReleaseSemaphore(IpcResources.hRoadSemaphores[algoritmo][i], 1, NULL) == 0) {
1017                     PERROR("PermisoAvanceCommit: ReleaseSemaphore 2");
1018                 }
1019             }
1020         }
1021     }
1022 }
1023
1024 void printCarretera()
1025 {
1026     int i, val;
1027
1028     val = IpcResources.nLotsArray[PRIMER_AJUSTE][0];
1029     for (i = 0; i < ROAD_LENGTH; ++i) {
1030         if (val != IpcResources.nLotsArray[PRIMER_AJUSTE][i]) {
1031             val = IpcResources.nLotsArray[PRIMER_AJUSTE][i];
1032             putchar(' ');
1033         }
1034
1035         printf("%d", IpcResources.nLotsArray[PRIMER_AJUSTE][i]);
1036     }
1037     putchar('\n');
1038 }
```